

©2000 Abaco PR, Inc. All rights reserved.

XML Open Object Model (XOOM)



XML Open Object Model

Table Of Contents

| | |
|---|-----------|
| INTRODUCTION | 5 |
| PRODUCT FEATURES..... | 5 |
| CLIENT FEATURES | 5 |
| SERVER FEATURES | 5 |
| XOOM CONCEPTS..... | 6 |
| ADMINISTRATION & CONFIGURATION | 6 |
| DEPLOYMENT | 6 |
| Web Server Files | 6 |
| XOOM Components | 6 |
| Application Files | 7 |
| Application Data | 7 |
| Version Control / Synchronization | 7 |
| CE INSTALLATION NOTES..... | 8 |
| WebServer | 8 |
| ADOCE 3.1..... | 8 |
| Pocket Internet Explorer Home Page | 8 |
| 128-Bit Encryption Pack | 8 |
| SECURITY | 9 |
| Initial System Access..... | 9 |
| Transfer Protocols | 9 |
| Server & Client Authentication..... | 9 |
| XOOM SCENARIOS | 9 |
| EXECUTION OF DEVICE SYNCHRONIZATION PAGE (ABDOSYNC.ASP) | 10 |
| EXECUTION OF SYNCHRONIZATION SERVER (ABSYNCSVR.DLL) | 10 |
| EXECUTION OF SOAP CALL BY ASP APPLICATION | 11 |
| ASYNCHRONOUS POST OBJECTS | 11 |
| SUPPORTED PLATFORMS | 11 |
| CLIENT PLATFORMS..... | 12 |
| SERVER PLATFORMS | 12 |
| CONNECTIVITY TO SAP R/3™ | 12 |
| INITIAL INSTALLATION | 13 |
| SYNCHRONIZATION PROCESS | 16 |
| CE Device Registry Entries | 20 |
| DEVELOPER COMPONENTS..... | 21 |
| Database Filter (abdbfilter.asp) | 21 |
| DEVICE COMPONENTS..... | 22 |
| ASP Extension (abaspex.dll) | 23 |
| Configuration Component (abconfig.dll) | 25 |
| Database Update Manager (abdbmgr.exe) | 26 |

| | |
|--|----|
| <i>Device Specific I/O: Magnetic Card Reader (abdevio.dll)</i> | 27 |
| <i>Device Specific I/O: Scanner (abdevio.dll)</i> | 28 |
| <i>Home Web Application</i> | 29 |
| <i>Information Component (abinfo.dll)</i> | 30 |
| <i>Monitoring Web Application</i> | 31 |
| <i>Remote Access Service Component (abRAS.dll)</i> | 32 |
| <i>Security Extension (absec.dll)</i> | 33 |
| <i>Shell Manager: Browser (abshell.dll)</i> | 34 |
| <i>Shell Manager: Shell (abshell.dll)</i> | 35 |
| <i>SOAP Component: Async (absoap.dll)</i> | 36 |
| <i>SOAP Component: Sync (absoap.dll)</i> | 38 |
| <i>Standard I/O: Printer (abstdio.dll)</i> | 39 |
| <i>Standard I/O: ScreenCapture (abstdio.dll)</i> | 40 |
| <i>Standard I/O: Serial (abstdio.dll)</i> | 41 |
| <i>Status Component (abstatus.dll)</i> | 42 |
| <i>Synchronization Client (abSync.exe)</i> | 43 |
| CLIENT DIRECTORY STRUCTURE | 46 |
| SERVER COMPONENTS | 47 |
| <i>Certificate Component (abcert.dll)</i> | 49 |
| <i>Connectory Proxy (abconnectorproxy.dll)</i> | 50 |
| <i>Console - Administration</i> | 51 |
| <i>Console - configuration</i> | 52 |
| <i>Console - monitoring</i> | 53 |
| <i>Console Utilities (abconsole.dll)</i> | 54 |
| <i>Cryptography Component (abcrypto.dll)</i> | 55 |
| <i>Database Schedule Executable (abdbschedule.exe)</i> | 56 |
| <i>Database Sync Packager (abdbsync.dll)</i> | 57 |
| <i>Information Server Agent (abinfosvr.dll)</i> | 58 |
| <i>Login Component (ablogin.dll)</i> | 59 |
| <i>Login Web Application</i> | 60 |
| <i>Packager Component (abpackager.dll)</i> | 61 |
| <i>SOAP Sync Agent (absoapsync.dll)</i> | 62 |
| <i>SOAP Server Client (abSOAPClient.dll)</i> | 63 |
| <i>Status Server Agent (abstatussvr.dll)</i> | 64 |
| <i>Synchronization Server (abSyncSvr.dll)</i> | 65 |
| <i>User Store Interface (IUser)</i> | 66 |
| SERVER DIRECTORY STRUCTURE | 67 |
| XML DOCUMENT SPECIFICATIONS | 69 |
| <i>Async Post XML document</i> | 70 |
| <i>Connector Interaction Document (abconnector.xml)</i> | 71 |
| <i>Data Package Document (abdatapackage.xml)</i> | 72 |
| <i>Data Update Document (dataupdate.xml)</i> | 73 |
| <i>Device Configuration Document (deviceconfig.xml)</i> | 75 |
| <i>Device Information document (info.xml)</i> | 78 |
| <i>Device Status document (status.xml)</i> | 81 |

| | |
|---|----|
| <i>Manifest Document (manifest.xml)</i> | 83 |
| <i>Printer Definition document ([printername]def.xml)</i> | 85 |
| <i>Printer Label document</i> | 87 |
| <i>User Store Interface Document (abusers.xml)</i> | 88 |
| <i>Xoom Registry Info Document (xoomreginfo.xml)</i> | 89 |

INTRODUCTION

XML Open Object Model (XOOM) is a suite of tools designed to provide enterprise application developers with the building blocks necessary to create fully featured enterprise applications that are delivered over the standard world wide web architecture.

Currently, the software development world is divided into two basic categories of developers and architectures. These two categories are Microsoft™ based technologies and Java™ based technologies. Enterprise customers typically standardize all of their operations on one of these two types of technologies. XOOM targets all enterprise customers with the goal of allowing developers to work with their favorite or policy-mandated technologies. In order to accomplish this, XOOM provides each of its components in two forms: one based on Microsoft™ technologies and one based on Java™ technologies.

PRODUCT FEATURES

Client Features

- Logic of the developed application will reside on the client device, enabling offline operation
- Developed application will be able to access objects native to client device
- Support for use of hardware extensions of client device while connected or disconnected
- Application developer will be responsible for checking the client state (connected vs. disconnected) when accessing databases etc..
- Developed application will be capable to cache HTTP post requests for subsequent asynchronous processing when client device regains network connectivity
- Client will be able to communicate with server securely, passing in and out of corporate firewall
- Support for client access to server-resident COM & Java Classes using SOAP technology

Server Features

- Server can be a standard internet web server; XOOM provides a solution for both Microsoft's IIS and the cross-platform web standard Apache.
- Server can utilize operating system of choice (Windows NT/2000, Linux, Unix)
- Support scaling of client instances via threading model through the use of COM+ components in Windows™ environments and Java Servlets in non-Windows environments, on the server side.
- Support processing of HTTP post requests received asynchronously from client applications
- Capability to remotely monitor client device statistics such as battery level, free memory space, installed applications and also remotely configure device

- Capable of applying a security policy which will be enforced throughout all aspects of the system's operation
- Server modules configured and administered through Web application which can be accessed remotely

XOOM CONCEPTS

Administration & Configuration

One component of the XOOM web services is the administrative component. The administrative component will be implemented with a Web application. The administrative component enables a system administrator to: create user accounts for XOOM clients, choose and configure a security policy for the XOOM system, choose the applications to be deployed to XOOM clients, and specify the configuration settings of XOOM client devices.

Deployment

XOOM applications are deployed to XOOM clients using the HTTP protocol. Applications are centrally stored on a web server and the entire XOOM framework is accessible through the web browser located on the client device.

The files comprising a complete XOOM installation fall into four categories: Web Server Files, XOOM Components, Application Files, and Application Data.

Web Server Files

For Windows Powered devices, the Web Server Files are the files necessary to run the Microsoft™ CE Web Server. These files are asp.dll, httpd.dll, httpdsvc.exe and httpdadm.dll. Also included in the file will be the files necessary to support 128-Bit encryption. The CE Web Server files are deployed in one CAB file which is compiled for the target processor. The CAB files for the CE Web Server are stored in the "WebServers" subdirectory of the XOOM installation directory on the server. The files are named for the processors that they support (i.e. "WebServerSH3.cab", "WebServerMIPS.cab") The CE Web Server will be deployed with a set of default settings. These settings can later be changed through the XML document that will be passed to the device during synchronize operations. The CAB file which installs the Web Server will set a registry entry which is flag indicating that the device should be reset. This flag will be read by other controls after synchronization to determine when the device must be reset.

XOOM Components

The XOOM components consist of the following files:

For Windows Powered devices, the XOOM components will be deployed in two CAB files. There will be one CAB file for the abdevio.dll which will be compiled for each supported device and there will be a second CAB file for the remaining XOOM components which will be compiled for each supported target processor. The CAB files for the XOOM components are stored in the "XOOMComponents" subdirectory of the XOOM installation directory on the server. This subdirectory has two subdirectories: "Device" and "Processor". The "Device" subdirectory has a folder for each supported device with the device specific CAB file (i.e. abdevio.dll cab) stored in that folder. The "Processor" subdirectory has a folder for each supported processor with the XOOM components corresponding to the processor. The CAB files for the processor-specific XOOM components are stored in the Processor subdirectory root and are named for the processors that they support (i.e. "XOOMComSH3.cab", "XOOMComMIPS.cab")

Application Files

For Windows Powered devices the application files will be deployed via a platform independent CAB file.

Each application can consist of several files (ex: .jpg's, .asp's, .htm's ...). Each application will have a directory in which all files comprising the application are stored. To deploy a new application or new version of an application a system administrator will use the XOOM-Admin to indicate the location of the application files and/or designate that these files have been updated. When this action is performed in the XOOM-Admin, a platform-independent CAB file will be created which will package all of the application files. This cab file will be located in the "Applications" subdirectory of the XOOM installation directory on the server and will be named according to the application name (i.e. "App1.cab", "App2.cab")

Application Data

The application data consists of an XML document which includes all database transformations corresponding to a given application and user. This XML document will be interpreted by the abdbmgr.exe component on the device in order to make the necessary database modifications. Each application data XML document will be stored in the corresponding user's directory in the "Users" subdirectory of the XOOM installation directory on the server.

For Windows Powered devices, the application data XML document will be deployed as a CAB file. This CAB file will be created by the XOOM Database Filtering component in the manner specified by the end user. When invoked, the Database Filtering component will perform the required database tasks and data preparation then update the corresponding XML document appropriately. The Database Filtering component will then package the XML document in a device independent CAB file.

Version Control / Synchronization

Version Control is the process of updating the applications that have been deployed to the device and if necessary the XOOM ActiveX controls. Version Control will be initiated by the user of the device as changes made to the device configuration during version control may necessitate a reboot of the device. The homepage/main menu application on the XOOM client device will have a "sync" button which will allow the user

to perform many functions including version control. When the "sync" button is clicked the following sequence of events takes place: 1- the device transmits any pending Asynchronous Post transactions; 2- the device performs version control of the applications; 3- the device performs database synchronization; 4- the device performs version control of the XOOM & Web Server components. Version control and synchronization of the applications, XOOM components and database will be initiated by the device using a SOAP request to the XOOM server. This SOAP request will include XML documents indicating the versions of applications and XOOM components currently installed on the device. Upon receipt of this request, the XOOM web server checks for any new versions of applications or components for the user and type of device sending the request and prepares a file for with any new versions for the device.

For Windows Powered devices, a CAB file will be sent to the device in response to the version control SOAP request. This CAB file will include any new applications for the user, any updated or new XOOM components for the device, and the database transformations necessary for the user.

CE Installation Notes

There are a number of issues, relevant to Windows CE, that the XOOM installation processes must handle:

WebServer

When web server is installed, webserver files will be installed with flags to indicate that the files should not be copied if an existing file is there. The Web Server will only be installed on initial installations. No attempts to upgrade an existing webserver will be made since, if it is installed in ROM memory it cannot be upgraded.

MSXML

The MSXML distribution will be installed with flags to indicate that the files should not be copied if an existing file is there. The MSXML component will only be installed on initial installations.

ADOCE 3.1

Most PocketPC devices are delivered with ADOCE 3.0. At time of writing, ADOCE 3.1 is the latest release of available for CE devices and XOOM installations on CE devices should include installation of the ADOCE 3.1 components.

Pocket Internet Explorer Home Page

HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main\StartPage = home page of internet explorer

128-Bit Encryption Pack

In order for the Pocket Internet Explorer to utilize 128-Bit encryption, XOOM must ensure that the Microsoft 128-Bit Encryption Pack is installed on the device.

Security

The XOOM web services allow a system administrator to establish a security policy that will be consistently enforced throughout all phases of execution of the system.

Initial System Access

The first option available to administrators of the XOOM system is whether users are allowed to initially access the system from beyond the local firewall. If the administrator chooses to prevent initial access to the system from beyond the firewall, devices which request the logon/authentication page from beyond the local area network will automatically be denied access.

Transfer Protocols

The XOOM system will support communication using both HTTP and HTTPS. The security policy in effect will dictate whether normal or secure posts will be used in transmissions which are controlled by the XOOM system. In the course of the application the developer has the option to use either HTTP or HTTPS as they see fit.

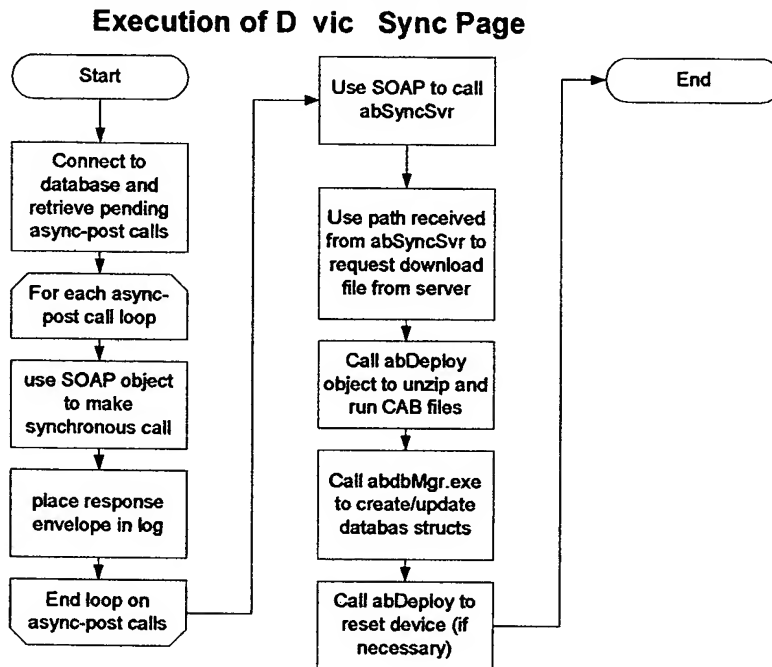
Server & Client Authentication

If mandated by the effective security policy, clients in the XOOM system will be required to have client certificates which can be authenticated by the local web server. After a user has logged-on to the XOOM system, the subsequent installation package that is sent to the client device will contain the client certificate assigned to the user/device by the web server. If the security policy enforces the use of SSL, each transmission between the client device and the server will include an authentication of the client certificate by the web server and an authentication of the server certificate by the client browser.

Through the administrative component of the XOOM services the system administrator is able to revoke client certificates if necessary and set the expiration time period for client certificates.

XOOM SCENARIOS

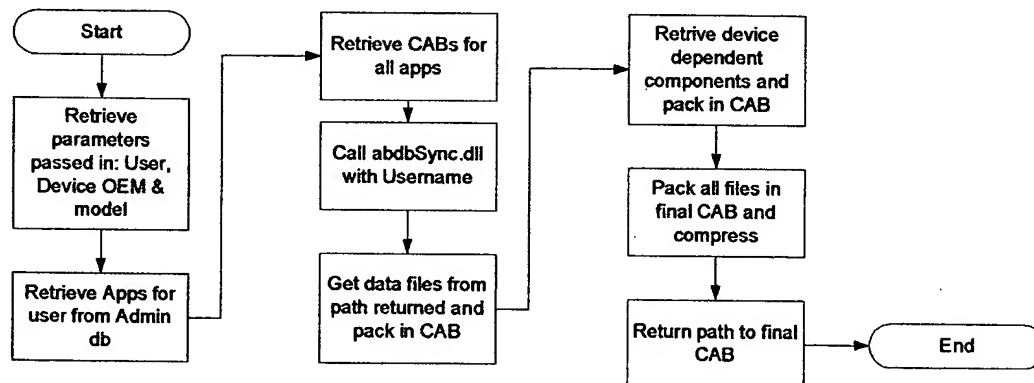
Execution of Device Synchronization Page (abdosync.asp)



Execution of Synchronization Server (absyncsvr.dll)

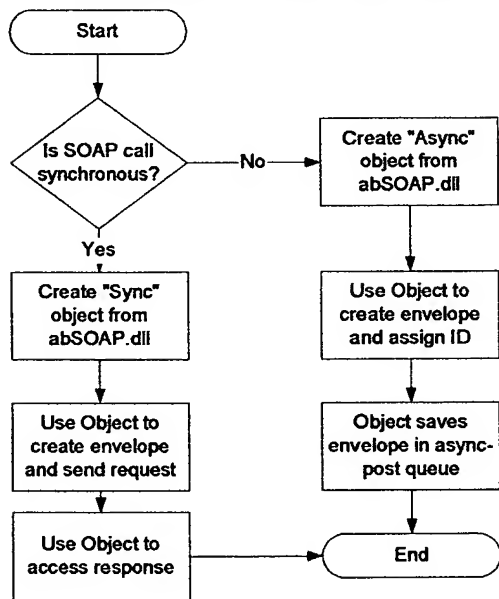
The Synchronization Server provides a method which can be called to retrieve XOOM applications and components which are not installed by the initial CAB file sent to the device during a new device installation. This method takes two parameters: The username of the user performing the install and the OEM-Model of the device in use.

Execution of Sync Svr (initial install)



Execution of SOAP Call by ASP Application

Execution of SOAP call from ASP



Asynchronous Post Objects

Asynchronous Post Objects will be called by the Synchronization Executive (abdosync.asp) on the device as queued SOAP requests are sent by the executive from the device to the SOAP router on the main server. The responses from these objects will be packaged in SOAP envelopes by the SOAP router and returned to the Synchronization Executive where the executive will deposit each return envelope in the log repository.

The format of the log repository is as follows:

| Application Name | Asynchronous Post ID | Response SOAP Envelope |
|------------------|----------------------|------------------------|
| | | |

The Application Name is the name of the application that made the asynchronous post SOAP call. The Asynchronous Post ID is an application defined ID which could possibly be used by an application to associate the log entry with the call or transaction to which the response corresponds. The Response SOAP Envelope holds the actual XML envelope received in response to the asynchronous call that was made.

SUPPORTED PLATFORMS

XOOM provides a cross-platform solution that supports devices and servers using the most popular operating systems in use across the world. XOOM aims to provide a

consistent development and administration framework for end-users allowing them to use the technologies with which they are most comfortable.

Client Platforms

| | <i>Windows CE</i> | <i>EPOC</i> | <i>Mobile Linux</i> |
|---------------------------|-------------------|------------------|---------------------|
| Browser | CE Web Server | Opera for EPOC | Opera for Linux |
| Local Server | CE Web Server | Java HTTP server | Java HTTP Server |
| Application Format | ASP & VB/J Script | JSP & JavaScript | JSP & JavaScript |
| Browser Extensions | ActiveX | JAR & Applets | JAR and Applets |

Server Platforms

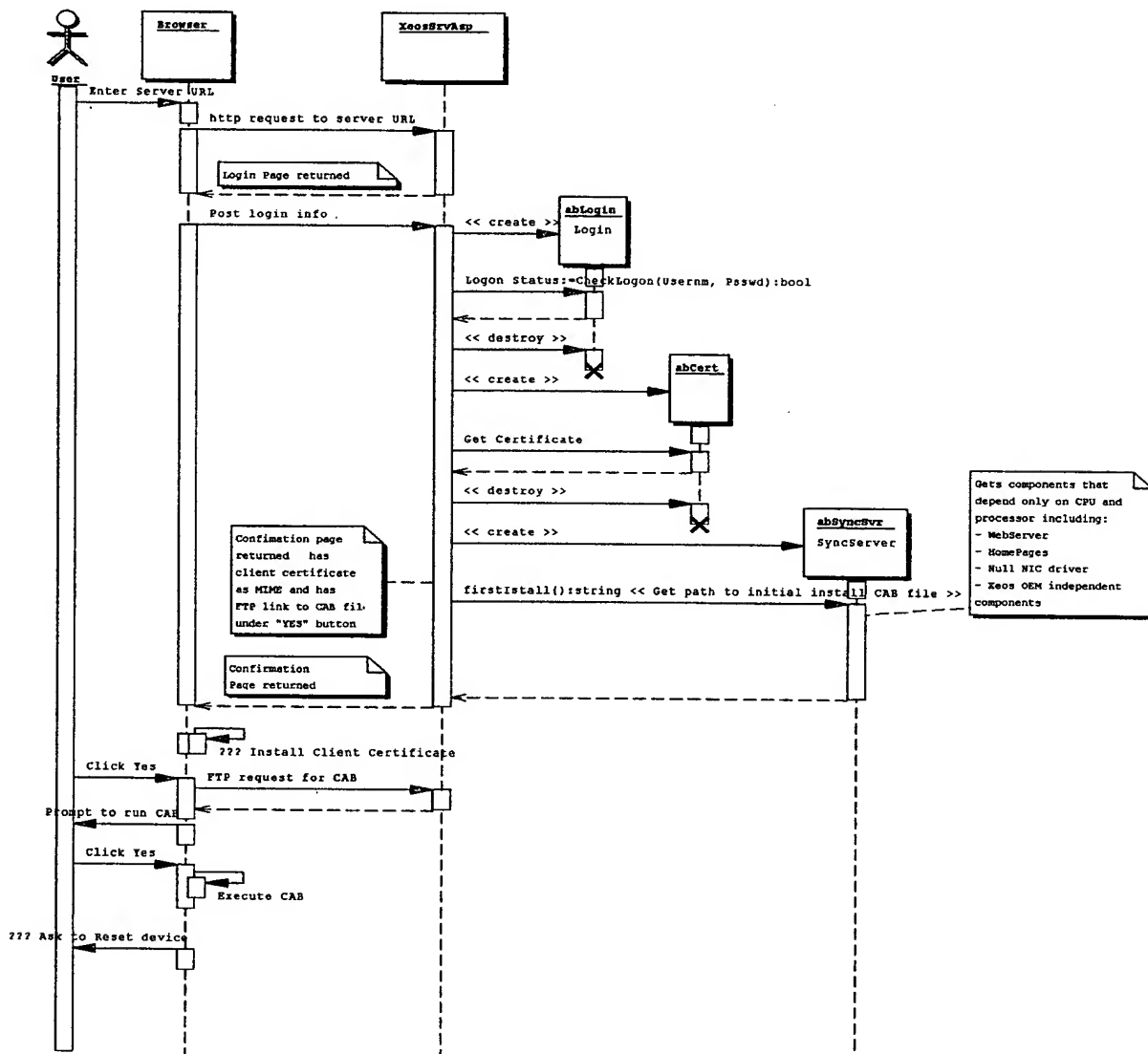
| | <i>Microsoft NT/2000</i> | <i>Unix</i> | <i>Linux</i> |
|-----------------------|--------------------------|-----------------|-----------------|
| Web Server | IIS | Apache | Apache |
| Servlet Engine | Tomcat | Tomcat | Tomcat |
| SOAP Engine | MS Soap toolkit | IBM Soap Devkit | IBM Soap Devkit |

CONNECTIVITY TO SAP R/3™

Due to its open architecture, XOOM allows a developer to use existing technologies such as the DCOM Connector, the Business Connector or the XML Message Server. In addition to these technologies, Abaco will provide a mechanism to call RFCs, BAPIs, and iDocs using a standard open format with proven scalability. The SOAP proxy to R/3 will provide the means by which RFCs, BAPIs, and iDocs can be called using SOAP XML messages. The SOAP proxy to R/3 will interface with Mocha™, Abaco's resource adapter for R/3 built using the J2EE specification to provide a robust, scalable connectivity solution for enterprise customers. Mocha™ provides managed connection pools, transaction encapsulation and a native SAP interface via JNI for connectivity to R/3 from remote clients. The SOAP Proxy to R/3 provides management of standard XML SOAP requests for conversion to SAP native RFCs, BAPIs and iDocs.

Initial Installation

Sequence 1: Initial Installation

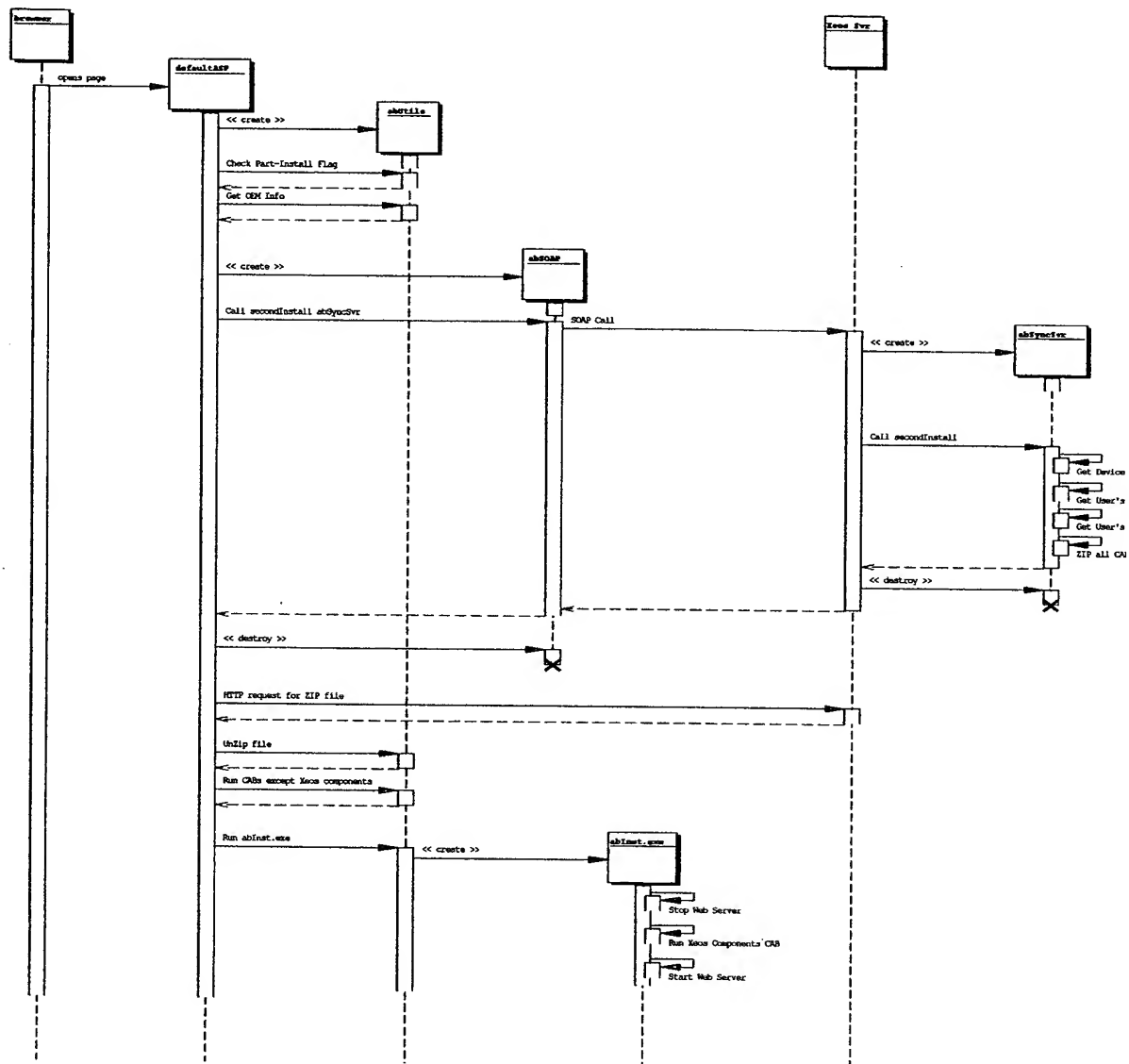


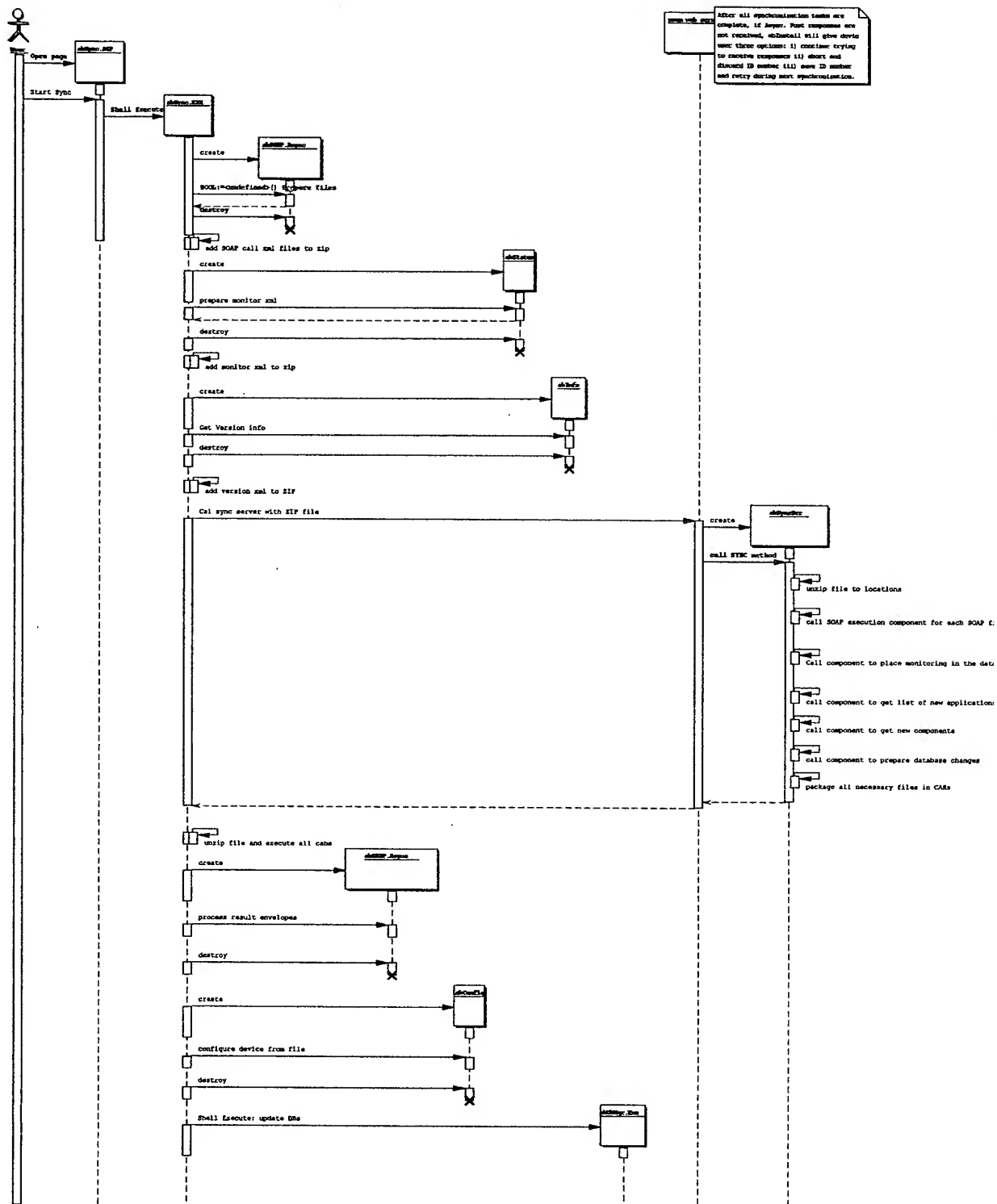
1. **The device user opens the web browser on the device and enters the URL of the central web server XOOM system login directory**
Using an input option provided by the device, the user enters the URL of the login directory. For secure, encrypted environments this directory should force usage of SSL.
2. **The web server with XOOM responds with a logon/authentication page**
The xoom login URL returns its default page which is a user authentication page prompting for a username and password. The security policy in effect may prevent this page from being sent in the case that the policy specifies that the page will only be delivered to client devices inside the firewall and a client requests it from outside the firewall.
3. **The client device operator submits the authentication page**
The user fills in a username and password and presses the submit button to return the completed form to the web server.

4. **The web server transfers to the abLogin page to validate the user** configured through the XOOM administration application and retrieves the processor/OS/version information from the client device's post. This information is in the UA-OS and UA-CPU parameters of the post header received from the device. The web server creates the client certificate for the device and sends the certificate along with a confirmation page.
5. **The server retrieves the files and the components for the users device platform and CPU**
esponds with a page containing a CAB file to install the CE Web Server and the XOOM components.
6. The CAB file, installs only the XOOM Components which are processor dependent, installs the local database system, installs the Web Server, installs the default XOOM ASPs, and sets the homepage for the Pocket Internet Explorer to the client default ASP application.
7. The Setup DLL completes the installation by rebooting the device. After the reboot, the device will open the web browser and load the default XOOM ASP page. This page will read a flag indicating that a new installation is in progress.
8. Upon reading the flag signifying a new installation, the application will invoke the server's Sync-Server object (abSyncSrv.dll) and call the method for completing new installations which returns the path to a CAB file containing the remaining items necessary to complete the installation.
9. The default XOOM ASP page will download and install the remaining items, which include the client applications, the device dependent XOOM components, and the application databases.

After completing the installation the default XOOM ASP application will return to the client home page which allows users to select an application, if multiple applications are deployed, or, if only one application is deployed the initial page of the application will be displayed.

Sequence 2: Initial Installation (PT 2 after reset)





1. **The device user selects the Sync option in the Xoom home-page application**
The Xoom Home-Page application's main menu displays a link to a synchronization page. When this link is selected the synchronization page is displayed which contains a button to activate the synchronization process. This button is pressed to start the synchronization process. Internally the page utilizes the abUtil component to open the abSync executable.
2. **The abSync executable retrieves the URL of the system's central web server**
Utilizing the abUtils control, abSync retrieves the URL of the central web server from local storage where it was placed upon installation of the Home-Page application
3. **The abSoap.Async control creates a file for each pending asynchronous post envelope**
The abSync executable calls a method of the Async object to convert each asynchronous post envelope, which is stored in a database table, into a text file. Each text file will be named using an ID number, also stored on the database, which corresponds to the post envelope. This ID number is a counter which is assigned to the post envelope by the Async object when the envelope is created to preserve the sequence in which the calls are made. Each file will have the file extension ".ase" to enable easy identification when extracted to a directory along with other file types. The text files are created in a directory which is passed in as a parameter to the method which prepares the files.
4. **The abStatus executable creates a xml document containing device monitoring information**
The abSync executable calls a method of the abStatus control which retrieves all predefined monitoring parameters from the device and packages them in a monitoring xml document which is returned as text. The abSync executable saves this text to a file with the filename "devstatus.xml".
5. **The ablInfo control creates an xml document containing component and application version information**
The abSync executable calls a method of the ablInfo control which retrieves version information about the Xoom components which are currently installed and the user applications which are currently installed. This information is inserted into a version xml document which is returned as text. The abSync executable saves this text to a file with the filename "devinfo.xml".
6. **The abSync executable retrieves a transaction identifier for the synchronization operation**
Using the abSoap control, the executable calls a method of the abSyncSvr control on the central web server to issue a unique transaction identifier for the synchronization operation in process. This transaction identifier is a Global Unique Identifier (GUID) which is created by the abSyncSvr control and returned. This identifier allows the synchronization operation to be encapsulated as a single transaction by the client and the server for easy referencing during subsequent stages of the process and for error recovery.
7. **The abSync executable zips all files associated with the synchronization**
The abSync executable uses the abUtils control to package all of the files needed by the central server into a zip file. The files included in the zip are all of the asynchronous post envelope files, the monitoring file and the version information file. The zip file is named using the unique transaction identifier obtained for the synchronization operation.
8. **The abStatus executable sends an HTTP post to the central server which deposits the zip file**
The abSync executable issues an HTTP post to the central web server in order to transmit the zip file containing all of the files for the synchronization operation. This file will be deposited in the "[Username]/Inbox" folder where "Username" is the name of the user which is using the device.

9. **The abStatus executable calls the main sync method of the abSyncSvr object**
The abSync executable calls a method of the abSyncSvr control to initiate processing of the synchronization information on the server. The executable passes the device identifier, and the transaction identifier as parameters to this method.
10. **The abSyncSvr object locates the zip file and extracts the contents**
The abSyncSvr object opens the zip file located in the directory where the abSync executable posted the file and extracts the contents.
11. **The abSyncSvr object calls abSOAPSvc to process all asynchronous post files**
The abSyncSvr object calls the abSOAPSvc control and passes the path to the location of the unzipped file as a parameter. The abSOAPSvc control filters the files in the specified path using the ".ase" extension. The abSyncSvr control subscribes to an event of the abSOAPSvc control which will be fired when all files have been processed.
12. **The abSOAPSvc component calls abSOAPClient for each async post call**
For each asynchronous post file encountered abSOAPSvc creates an instance of the abSOAPClient control to execute the SOAP call contained in the ".ase" file. The abSOAPClient control fires an event when the response to the call is received. This event is attended by the abSOAPSvc control. When all ".ase" files have been processed in this manner the abSOAPSvc control fires an event indicating that all files have been processed.
13. **The abSyncSvr object submits the monitoring information to the abStatusSvr component**
The abSyncSvr object passes the path where the files have been unzipped along with the unique device identifier to the abStatusSvr object. The abStatusSvr object searches for the monitoring file "devstatus.xml" and reads the monitoring information from the document.
14. **The abStatusSvr object saves the monitoring information to the server repository**
Using the monitoring information read from the document, the abStatusSvr object overwrites the monitoring information for the specified device with the new information in the system database. The abStatusSvr control requests the device configuration xml file associated with the device ID. If configuration changes are pending from the Console, this xml file will be included in the package deployed to the client device.
15. **The abSyncSvr object submits the version information to the abInfoSvr component**
The abSyncSvr component calls a method of the abInfoSvr control, passing the path to the version file, the unique id of the device, and the path in which cab files should be placed. The abInfoSvr control retrieves the "devinfo.xml" from the specified path and reads the version information.
16. **The abInfoSvr control retrieves CAB files of new components and applications**
The abInfoSvr component uses the abConsole control to determine if the version of components and applications installed are in need of update. The abInfoSvr component uses the device id to determine the applications which are currently assigned to the users group. If any existing applications are out of date or if any new applications are assigned, abInfoSvr requests the new CAB files of the applications. If an application is no longer assigned to the user the abInfoSvr adds the application to an "appdelete.xml" file which will be interpreted on the device to delete applications. If a new version of the installed components exists on the server the control requests the CAB files of the new components.
17. **The abSyncSvr object calls the abDataSync control to prepare the application database transformation file**
The abSyncSvr object will check the Console settings to see if the asynchronous SOAP calls must be completed before the database synchronization process must start. If this option is not active, the abSyncSvr object will immediately call a method of the abDataSync control, with

the device id as a parameter, to prepare the application database changes. If the Console option indicates that the SOAP calls must be completed, abSyncSvr will only call abDataSync after the event fires indicating that the asynchronous SOAP calls are completed.

18. **The abDataSync control retrieves the pending database updates for the device user**
19. **The abSyncSvr object packages all files prepared for the user in a zip file**
The abSyncSvr component adds all CAB files and xml files to a zip file which is named using the transaction identifier for the synchronization operation.
20. **The abSync executable checks to see if the synchronization processes are complete on the server**
The abSync executable on the device calls a method of the abSyncSvr object to check if the server tasks are complete and a response zip file is ready. The executable passes the transaction identifier for the synchronization operation as a parameter to the method. If the response file is ready abSyncSvr responds with the path to the zip file requested.
21. **The abSync executable retrieves the response zip file from the server**
The abSync executable issues an HTTP get to the server requesting the file in the path returned by abSyncSvr.
22. **The abSync executable extracts the contents of the zip file to a temporary directory**
The abSync executable uses the abUtils control to extract the contents of the zip file to a directory on the client device.
23. **The abSOAP.Async object processes the response envelope files**
The abSync executable uses the abSOAP.Async control to process the asynchronous post response envelope files which were extracted from the zip file. The executable calls a method of the Async object passing the path to the unzipped files as a parameter. The Async object filters the files in the directory using the ".ase" file extension and processes each response envelope encountered.
24. **The abDBMgr executable performs the database transformations**
The abSync executable calls the abDBMgr executable which searches for the database configuration xml file extracted from the response zip file and performs the necessary transformations on the application databases.
25. **The abSync executable deletes any obsolete applications**
The abSync executable reads the "appdelete.xml" file to retrieve the obsolete applications and deletes the applications from the device.
26. **The abConfig object makes configuration changes to the device settings**
The abSync executable calls the abConfig control and passes it the path to the extracted device configuration xml file. The abConfig control processes the information in the configuration file and performs the specified device configurations
27. **The abSync executable stops the Web server and prompts the user to reset the device**
The abSync executable calls the web server utilities executable to stop the web server. The abSync executable displays a message to the user indicating that the device should be rebooted. The executable attempts to reboot the device automatically.

Title: CE Device Registry Entries

Date Last Revised:

Authors:

Purpose:

HKCR – HKEY_CLASSES_ROOT
HKCU – HKEY_CURRENT_USER
HKLM – HKEY_LOCAL_MACHINE

Entries:

Path: HKCU\Software\Abaco\XOOM\PartialInst Data Type: DWORD (0x00010001)

Indicates that an installation of XOOM components is in progress. Used by ASP pages on device to determine whether installation process should be continued or regular operation should resume.

Values: 1- True (Installation in progress); 2- False (Installation complete)

Path: HKLM\Software\Apps\%AppName%\XOOMApp Data Type: DWORD (0x00010001)

Indicates that an installed application is part of a XOOM package. Used by utility control to alter application settings so that re-installation of application does not prompt for acceptance.

Values: 1-True(application is part of XOOM)

Methods:

Name:

Return Type:

Parameters:

DEVELOPER COMPONENTS

Title: Database Filter (abdbfilter.asp)

Date Last Revised:

Authors:

Purpose:

The Database Filter provides a tool for system administrators and developers to create a client database structure for a XOOM application and to map tables and fields for the client database from a central server resident database. The Database Filter also provides the mechanism by which a database package for a XOOM application can be linked to an accompanying data filtering object which has been implemented, by the application developer, to perform application and user specific transformations on data extracted by the database filter from the larger central system database.

When executed, the database filter will create an XML document representing the table mappings, relationships, and filters that have been created with the tool. This XML document can be later used by a data filtering object created by the developer to prepare application data for a specific user.

DEVICE COMPONENTS

The XOOM device components extend the capabilities of the devices web server and browser and also facilitate many XOOM specific operations on the client device.

| | |
|--|---|
| ASP Extension (abaspex.dll) | Provides extensions to ASP capabilities of standard CE web server |
| Configuration Component (abconfig.dll) | Reads deviceconfig.xml document and makes appropriate changes to device configuration |
| Database Update Manager (abdbmgr.exe) | Interprets XOOM database transformation XML files and executes locally on device |
| Device Specific I/O: Magnetic Card Reader (abdevio.dll) | Provides object for device specific peripherals such as scanner and magnetic card reader |
| Information Component (abinfo.dll) | Provides current versions etc. of XOOM components and installed applications |
| Synchronization Client (abSync.exe) | Embedded VB app which manages the sync process from the client side. |
| Security Extension (absec.dll) | Provides ISAPI extension for XOOM specific security of local web server |
| Shell Manager: Browser (abshell.dll) | Provides interface to modify operating system shell and change the browser interface |
| SOAP Component: Async (absoap.dll) | Provides objects to execute synchronous and asynchronous SOAP calls |
| Status Component (abstatus.dll) | Provides current status of device such as battery level, available memory |
| Standard I/O: Printer (abstdio.dll) | Provides objects for cross-platform input/output options such as serial, printer and touch-screen capture |
| Remote Access Service Component (abRAS.dll) | Provides interface to telephone API of device |
| abUtils.dll | Provides deployment support functionality such as resetting of device and unzipping of deployed files |

Interfaces:

Device components implement the following interfaces:

IErrors – allow errors to be captured in scripting error object

IOjectSafety – allow controls to be embedded in HTML pages without warning message to user

Title: ASP Extension (abaspex.dll)

Date Last Revised:

Authors:

Purpose:

The ASP Extension component provides the normally available ASP Application object, which is not currently provided by the Microsoft™ CE ASP engine. The Application object provides a means of saving scalar values which can be accessed at any point within an ASP application. The data saved using the application object is maintained between user sessions and soft-resets of the CE device. The application object contains a collection called **Contents** which is used to manage the values stored for the application.

PropertiesName: **AppName**Data Type: **BSTR***

Read/Write property. If an application object has already been created using the string name provided, AppName returns a reference to the existing application object. If no object was previously created with the name, a new object is created.

Name: **Contents.Count**Data Type: **Long***

Read property. Provides a total of the number of items stored in the contents collection.

Name: **Contents.Item(key)**Data Type: **Variant**

Read/Write Property. *Key* is a string identifier for the value which is saved. When read, this property returns the value currently associated with *key*. When written to, the value currently associated with *key* is overwritten with the new value provided. If *key* is not defined, *key* is created and the value provided is assigned.

Methods:Name: **Lock**

Return Type:

Parameters: **none**

Prevents other user sessions from writing to the Contents collection

Name: **Unlock**

Return Type:

Parameters: **none**

Opens the Contents collection to write access from other sessions. Called after a previous call to **Lock()**.

Name: **Contents.Remove(key)**

Return Type:

Parameters: **Key:BSTR***

Removes the item associated with *key* from the Contents collection.

Name: **Contents.RemoveAll()**

Return Type:

Parameters: none

Removes all items from the contents collection.

Examples:

VBScript:

```
Dim Application
Set Application = CreateObject("ABASPEX.Application")
Application.AppName = "VirtualDirectory"
Application.Lock
If (Application("Visits") <> Empty) Then
    Application("Visits") = Application("Visits") + 1
Else
    Application("Visits") = 1
End If
Application.Unlock
Set Application = Nothing
```

Title: Configuration Component (abconfig.dll)

Date Last Revised:

Authors:

Purpose:

The Configuration component provides an interface to make configuration changes to the device operating system and to retrieve the current configuration settings. The Pocket Internet Explorer defaults with Error messages for scripting turned off. Changing this setting is possible through the device configuration interface.

Properties

Name:

Data Type:

Name:

Data Type:

Methods:

Name: Get

Return Type: BSTR*

Parameters:

Returns the current configuration settings of the device in an XML document

Name: Set(Settings)

Return Type:

Parameters: Settings: BSTR*

Configures the device according to the XML document passed as a string in the Settings parameter.

Title: Database Update Manager (abdbmgr.exe)

Date Last Revised:

Authors:

Purpose:

The Database Manager performs transformations against the local client database as prescribed by the XML document delivered to the device during synchronization operations.

Properties

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: Device Specific I/O: Magnetic Card Reader (abdevio.dll)

Date Last Revised:

Authors:

Purpose:

The Device I/O component provides programmatic access to hardware input/output peripherals such as scanners and magnetic card readers which utilize manufacturer specific APIs. A distinct version/compilation of this component will exist for each supported device.

Abdevio provides two objects "Scanner" and "MSR". If a particular device does not support either object the object will not be included in the DLL for that device. A programmer can check for this by attempting to create the object and catching the error if the component cannot create the object.

In the pocket Internet Explorer the value input through the hardware will be set to the control currently with the focus if the control is a TextBox, ListBox, or ComboBox. In the full internet Explorer and Embedded Visual Basic or C++ applications (which support ActiveX control events) and event will be triggered which can be used to assign the input to any desired screen element or otherwise manipulate the input.

Properties

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Name:

Return Type:

Parameters:

Title: Device Specific I/O: Scanner (abdevio.dll)

Date Last Revised:

Authors:

Purpose:

See **Device Specific I/O: Magnetic Card Reader (abdevio.dll)** for further description.

Properties**Name:** Enable**Data Type:** Boolean

Read/Write Property. When set to True, the current symbology is activated in the scan engine. When set to False, the scan engine is disabled.

Name: Symbology**Data Type:** Long

Read/Write Property. Sets the current symbology of the scanner object to the barcode symbology corresponding to the long value assigned. Note: the symbology will not be actually set until the **Enable** property is assigned a true value.

Name: UseHWEvent**Data Type:** Boolean

Read/Write Property. Specifies whether the object will fire the hardware event when information is read by the scanner.

Methods:**Name:** Trigger**Return Type:****Parameters:** none

Executes a read operation by the scanner hardware.

Title: Home Web Application

Date Last Revised:

Authors:

Purpose:

The Default Application provides the default interface

Properties

Name: abapps.asp

Data Type:

Listing of available applications with links to start each application

Name: appfinder.asp

Data Type:

Name: default.asp

Data Type:

The first time the default application is run, it checks a flag to see if the device is a new device which has just installed the XOOM components. If the device is new, it immediately runs the Synchronization Executive to install the users applications and supporting databases.

Name: dosync.asp

Data Type:

Page called when 'sync' button is pressed which executes all synchronization tasksThe synchronization executive is called when the synchronization button is activated on the Synchronization Application page. The Synchronization Executive performs two main tasks: 1 – executes pending asynchronous post calls; 2 – calls the Synchronization Server to perform (i) application version control, (ii) synchronization of user data, and (iii) XOOM component version control.

Name: sync.asp

Data Type:

The Synchronization page with info time of last sync and button to initiate sync operationSynchronization Application provides a button labeled "sync" which is used to execute a device synchronization operation. When the "sync" button is pressed the Synchronization Executive is called to execute the synchronization operation.

The Synchronization Application also provides a total of the pending asynchronous post operations and the date and time of the most recent synchronization operation.

Title: Information Component (abinfo.dll)

Date Last Revised:

Authors:

Purpose:

The Device Information component is used to retrieve the versions of the applications installed and the versions of the XOOM components installed. These versions are available to this component from the "manifest" XML documents which accompany each application and the XOOM components.

Properties

Name:

Data Type:

Name:

Data Type:

Methods:

Name: SystemVersion

Return Type:

Parameters:

Returns version information of XOOM components on the device in an XML document

Name: AppVersion

Return Type:

Parameters:

Returns version information on all installed applications in an XML document

Title: Monitoring Web Application

Date Last Revised:

Authors:

Purpose:

The Monitoring Application allows the current status of the device to be viewed remotely when the device is 'online'.

Pages

Name: monitor.asp

Data Type:

Page which provides status information regarding device

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Title: Remote Access Service Component (abRAS.dll)

Date Last Revised:

Authors:

Purpose:

Provides ability to open a RAS connection, check if an existing connection is already open and close a connection. Administrator has the ability to create a RAS Connection Entry using the device configuration section of the Console.

A connection opened with an instance of the abRas component will remain open after the instance is destroyed. A reference to a previously opened connection can be obtained from a new instance of the abRAS component and can be disconnected using the hang-up method.

Properties

Name:

Data Type:

Name:

Data Type:

Methods:

Name: Dial(*Entryname*)

Return Type: BOOL

Parameters: Entryname: BSTR

Opens a RAS connection using the phonebook entry specified in the Entryname parameter. If a different phonebook entry currently has the active connection, this connection is ended and a new connection is opened using the phonebook entry specified. If a connection is already active using the specified phonebook entry the method takes no action. True is returned when the connection is activated; False is returned if the method is unable to open the connection.

Name: HangUp(*Entryname*)

Return Type:

Parameters: Entryname: BSTR

If the phonebook entry specified in the parameter Entryname is currently connected, this method ends the connection.

Name: IsConnected(*Entryname*)

Return Type: BOOL

Parameters: Entryname: BSTR

Returns True if the phonebook entry specified in the parameter Entryname is currently connected or False if the phonebook entry is not currently connected.

Title: Security Extension (absec.dll)

Date Last Revised:

Authors:

Purpose:

The Security Extension will provide security to the CE web server. Using this ISAPI filter the server will be instructed to discard all request except those coming from localhost or the central server.

Properties

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: Shell Manager: Browser (abshell.dll)

Date Last Revised:

Authors:

Purpose:

The Shell Manager will allow applications to modify the current state of both the user interface and the Pocket Internet Explorer.

The Browser object, provided by the Shell Manager, allows an application to show or hide various elements of the Pocket Internet Explorer user interface.

Properties

Name: LockAppKeys(*Lock*)

Data Type: BOOL

Prevents or Enables use of hardware buttons as shortcuts to device applications. If a True value is passed with the *Lock* parameter the device buttons do not launch their assigned applications; If a False value is passed the buttons revert to default operation.

Name:

Data Type:

Methods:

Name: ShowAddressBar

Return Type:

Parameters: none

Toggles the visible state of the AddressBar of the Pocket Internet Explorer. If the AddressBar is visible, a call to this method will hide the AddressBar. If it is not visible, a call to this method shows the AddressBar.

Name: ShowMenuWorker(*show*)

Return Type:

Parameters: Show: Boolean

Shows or Hides the Internet Explorer Menu Bar which provides access to the Menus and Buttons of the Pocket Internet Explorer. If a True value is passed with the *Show* parameter the Menu Bar is shown; If a False value is passed the Menu Bar is hidden.

Title: Shell Manager: Shell (abshell.dll)

Date Last Revised:

Authors:

Purpose:

The Shell object, provided by the Shell Manager, allows an application to show or hide various elements of the default Windows CE user interface.

Properties

Name:

Data Type:

Name:

Data Type:

Methods:Name: ShowSipButon(*show*)

Return Type:

Parameters: Show: Boolean

Shows or Hides the Soft Input Panel icon which is normally displayed on the taskbar of the Windows CE user interface. If a True value is passed with the *Show* parameter the SIP icon is shown; If a False value is passed the icon is hidden.

Name: ShowStartIcon(*show*)

Return Type:

Parameters: Show: Boolean

Shows or Hides the Start icon which provides access to the Start menu on Windows CE. If a True value is passed with the *Show* parameter the Start icon is shown; If a False value is passed the icon is hidden.

Name: ShowTaskBar(*show*)

Return Type:

Parameters: Show: Boolean

Shows or Hides the Taskbar which is normally displayed on the 'Today' screen of Pocket PC. If a True value is passed with the *Show* parameter the Taskbar is shown; If a False value is passed the Taskbar is hidden.

Title: SOAP Component: Async (absoap.dll)

Date Last Revised:

Authors:

Purpose:

The SOAP component allows applications to create, send and receive SOAP envelopes to facilitate execution of methods of remote objects which do not reside on the device.

The AsyncPost Object will provide a similar interface as the SyncPost object for creating a SOAP envelope but will store all envelopes create in local storage where they will remain until the next user-initiated synchronization operation.

Properties

Name: BodyExists (read only)

Data Type: Variant

Checks if a BODY element is associated with the current envelope

Name: CoerceResults

Data Type: Variant

Indicates whether result parameters should be converted to appropriated datatypes. If false all results are left as strings.

Name: ErrorOccurred (read only)

Data Type: BOOL

Name: FaultCode (read only)

Data Type: BSTR

Name: FaultString (read only)

Data Type: BSTR

Name: Headers

Data Type:

The collection of SOAP envelope Header parmeters associated with the current envelope

Name: LastErrorCode (read only)

Data Type: Long

Error code associated with the error, if any, that occurred during the last execution

Name: LastErrorMsg (read only)

Data Type: BSTR

Error message, if any, from the last call SOAP envelope executed

Name: MethodName

Data Type: BSTR

Name of the method to be called by the SOAP envelope

Name: ObjectName

Data Type: BSTR

Name of the class exposing the method to be called by the SOAP envelope

Name: Parameters

Data Type:

The collection of parameters associated with the SOAP envelope

Name: RequestText (read only) Data Type: BSTR

Name: RequestXML Data Type:

Name: ResponseText (read only) Data Type: BSTR

Name: ResponseXML Data Type:

Name: Result (read only) Data Type: Variant

Name: RouterURL Data Type: BSTR

The URL to which the SOAP envelope will be sent when the Execute method is called

Name: SOAPFaultExists (read only) Data Type: Variant

If true, a fault is present in the current response envelope.

Methods:

Name: Execute Return Type: Long

Parameters: none

Send SOAP Request to URL specified in 'RouterURL' parameter

Name: Reset Return Type: Void

Parameters: none

Resets the control to the default SOAP template

Title: **SOAP Component: Sync (absoap.dll)**

Date Last Revised:

Authors:

Purpose:

The SyncPost Object enables the creation and transmission of SOAP envelopes and retrieval of responses to transmitted requests.

Error Codes:

XS_OK = 5000

XS_SOAP_FAULT = 5001

XS_NO_ROUTE = 5002

XS_DNS_ERROR = 5003

XS_UNSPECIFIED_ERROR = 5004

Properties

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: Standard I/O: Printer (abstdio.dll)

Date Last Revised:

Authors:

Purpose:

The Standard I/O component provides access to hardware functionality that is not manufacturer specific. Functionality such as: access to the serial port, capture of touch-screen input, and access to portable printers is provided in the Standard I/O component.

The Printer object allows applications to print to various models of portable printers using the serial port or the infrared port.

In the pocket Internet Explorer the values input through printers capable of receiving input, will be set to the control currently with the focus if the control is a TextBox, ListBox, or ComboBox. In the full internet Explorer and Embedded Visual Basic or C++ applications (which support ActiveX control events) and event will be triggered which can be accessed through code to set the desired values and otherwise manipulate the input.

Properties

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: **Standard I/O: ScreenCapture (abstdio.dll)**

Date Last Revised:

Authors:

Purpose:

Properties

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: **Standard I/O: Serial (abstdio.dll)**

Date Last Revised:

Authors:

Purpose:

Properties

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: Status Component (abstatus.dll)

Date Last Revised:

Authors:

Purpose: The Device Status component is used by the XOOM monitoring application to retrieve the

The Device Status component is used by the XOOM monitoring application to retrieve the current running status of the device.

Properties The Device Status component has the following properties:

Name:

Data Type:

Name:

Data Type:

Methods: The Device Status component has the following methods:

Name: Get

Return Type: BSTR*

Parameters:

Returns the current status of the device in an XML document; includes battery level, free memory etc...

Title: Synchronization Client (abSync.exe)

Date Last Revised:

Authors:

Purpose: The installation executive is used by system applications to stop the web server, run an

executable file, then restart the web server. The primary use of the Installation Executive is by the system application which completes an initial installation or synchronization operation. This system application must install system components which may be in use by the web server and restart the web server to activate changes made to the web server configuration. The Installation Executive is used by this system application to first stop the web server; second run a CAB file to install system components; and third restart the web server. A separate executable is required to accomplish this task since the system application runs in the same thread of execution as the web server. Stops Web server and runs CAB file for XOOM components

Parameters

Name: File

Data Type: BSTR

The Installation executive will run the file specified by the file argument after stopping the web server. If no file is specified, the Installation executive will stop and start the web server without running a file in-between.

Name:

Data Type:

Name:

Data Type:

Title: Utility: (abUtils.dll)

Date Last Revised:

Authors:

Purpose:

The Utility component provides the capability to perform functions typically reserved for native applications. The primary goal of the Utility component is to enable various installation functions to be scripted via ASP pages. However the end developer is free to utilize the component to perform actions in client applications if desired.

Properties

Name:

Data Type:

Methods:Name: AcceptPKCS7(*Certificate*)

Return Type: none

Parameters: Certificate: VARIANT;

Installs the client certificate passed in the *Certificate* parameter into the local certificate store

Name: GetServerURL

Return Type: BSTR*

Parameters:

Returns the value stored in the registry key for the system server address. This value is set during the installation process.

Name: IsPartialInst

Return Type: Variant

Parameters: none

Returns the current installation status of the device. If the device is in between the start of an initial installation and the completion of the initial installation, this method returns True. If initial installation of the framework is complete on the device, this method returns false.

Name: SaveFileToDisk(*Filename, FileData*)

Return Type: Long*

Parameters: Filename: BSTR; FileData: VT_ARRAY, FileLen: Long*

Creates a file with the name specified in the *Filename* parameter and writes the data contained in the *FileData* parameter into the newly created file. If a file already exists at the specified location with the same name, it is overwritten. The *Filename* parameter should include the path and the filename. The size in bytes of the file created is returned by the method.

Name: ShellExec(*Filename, Params*)

Return Type:

Parameters: Filename: BSTR; Params: BSTR

Runs the executable file specified in the *Filename* parameter and passes the string specified in the *Params* parameter as command line arguments when calling the executable. The *Filename* parameter should include the path and the filename.

Name: `ReadDiskFile(Filename)`

Return Type: VARIANT

Parameters: *Filename*: VARIANT;

Reads text information from file specified in *Filename* parameter. Adjusts for differences between files saved in ASCII and Unicode formats.

Name: `UnzipFile(Filename, DestDir)`

Return Type:

Parameters: *Filename*: BSTR; *DestDir*: BSTR

Decompresses the file specified in the *Filename* parameter and saves the decompressed contents to the path specified in the *DestDir* parameter. The *Filename* parameter should include the path and the filename.

SERVER COMPONENTS

The server controls which will be heavily utilized as the number of system users increases will be implemented using COM+. Use of COM+ provides support for object pooling and events between objects. The implementation of the objects to handle asynchronous post envelopes on the server will rely heavily on the COM+ event services.

| | |
|--|---|
| Certificate Component (abcert.dll) | Retrieves certificate and assigns to user |
| Connectory Proxy (abconnectorproxy.dll) | Provides interface to XOOM Connector from COM |
| Console - Administration | Web application to manage and configure system |
| Console Utilities (abconsole.dll) | Provides interface to retrieve information from Console repository |
| Cryptography Component (abcrypto.dll) | Provides capability to encrypt and decode text information |
| Database Schedule Executable (abdbschedule.exe) | Prepares application database transformation files for all users for all applications assigned to run using a schedule. |
| Database Sync Packager (abdbsync.dll) | Prepares application database transformation file for a specified user |
| Information Server Agent (abinfosvr.dll) | Retrieves updated versions of components and applications during synchronization |
| Login Component (ablogin.dll) | Provides login services during initial logon process |
| Login Web Application | Initiates installation process on during initial logon process |
| Packager Component (abpackager.dll) | Packages applications, and files into CAB deployment files |
| SOAP Sync Agent (absoapsync.dll) | Processes batch of incoming asynchronous post files |
| SOAP Server Client (abSOAPClient.dll) | Executes a SOAP call from the central server during synchronization |
| Status Server Agent (abstatussvr.dll) | Records status information into Console data repository |
| Synchronization Server (abSyncSvr.dll) | Manages entire synchronization process on central server |

**User Store Interface
(IUser)**

Provides interface to user repository for Console

Title: **Certificate Component (abcert.dll)**

Date Last Revised:

Authors:

Purpose:

The Certificate Component is used as an interface between the system login application and a digital certificate server.

Properties:

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Methods:

Name: GetCertificate(*User*)

Return Type:

Parameters: User: BSTR*

Gets a certificate for a given user from a certificate server

Title: **Connectory Proxy (abconnectorproxy.dll)**

Date Last Revised:

Authors:

Purpose:

Provides connectivity to ERP and other enterprise backend systems via an object model implemented using the Java 2 Enterprise Edition specifications for resource adapters.

Properties

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: Console - Administration

Date Last Revised:

Authors:

Purpose:

The Console provides a means of configuring and administering a XOOM installation via a web interface. The Console asp files utilize relative paths for links between pages.

In order to utilize efficient design and to provide administrators with a simple yet flexible interface the Console will utilize the directory structure of the XOOM system installation along with systematic use of XML documents to store any required information. The XOOM system directory structure includes folders for each user, each device type and each application which will be used somewhat like an LDAP repository to store relevant information.

Properties

Name: User Repository

Data Type:

The Console will utilize a control to validate user information to allow this information to reside in systems other than the XOOM system database. A setting in the Console application specifies which user repository will be used in the system. The Console supports the default XOOM user store (abuser_xoom.dll), Active Directory (abuser_adsi.dll), and SAP R/3 (abuser_sapr3.dll). Depending on the setting the console will use the appropriate control to validate the user information. IXoomUsers

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: Console - configuration

Date Last Revised:

Authors:

Purpose:

Entry Name maxlen 20

-Phone Number maxlen 128

-UserName maxlen 256

-Password maxlen 256

-Domain maxlen 15

The Remote Device Monitoring option of the Console will provide the option to view monitoring information for devices, saved during each devices last sync, or to single out a particular device and attempt to get up to date monitoring information on that specific device.

The Console will not provide interfaces to configure the security policy or certificate settings of the central server.

Properties

Name: Secure Synchronization

Data Type:

Specifies whether the synchronization process should utilize SSL for communications between the device and the server. This setting is saved to the registry xml file stored on the device.

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: Console - monitoring

Date Last Revised:

Authors:

Purpose:

The Monitoring area of the Console application allows a system administrator is able to view status information of client devices in real-time or the most recent information reported. The monitoring service allows an administrator to determine which users are connected, view the battery and available memory levels of the connected devices and view statistical information about the device and user such as the time elapsed since their last communication with the server and the versions of applications and XOOM components that they have installed.

Properties

Name:

Data Type:

Name:

Data Type:

On the client ASP pages running in the CE web server respond to requests with status information regarding the device.

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: Console Utilities (abconsole.dll)

Date Last Revised:

Authors:

Purpose:

Provides utilities for extracting information from the Consoles repository of system information. Primarily used during the synchronization process to retrieve information pertaining to the user executing the sync.

Properties

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: **Cryptography Component (abcrypto.dll)**

Date Last Revised:

Authors:

Purpose:

Provides capability to encrypt information which will be stored on the XOOM server. Primarily used by the Console to encrypt database password information in datasource definition files.

Properties

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: **Database Schedule Executable (abdbschedule.exe)**

Date Last Revised:

Authors:

Purpose:

The Database Schedule Executable is an executable file that, when called, executes data preparation for a given application for all registered users.

Properties

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: **Database Sync Packager (abdbsync.dll)**

Date Last Revised:

Authors:

Purpose:

The Database Synchronization Object provides methods which create a client database transformation file which will subsequently be downloaded to a device. When called, the Database Synchronization object opens a table created by the Console which lists the Database Packages which must be run for a given application and whether these database packages are to be run by *abrundbsync.exe* on a schedule or run dynamically as each user synchronizes.

The Database Synchronization Object will provide a method which will be called by *abrundbsync.exe* which retrieves all of the users from a table created by the Console and for a given application creates all of the database packages which are to be run on the schedule for each user.

The Database Synchronization Object will provide a method which allows the Synchronization Server to pass a username as a parameter. This method will retrieve all of the applications assigned to the given user from a table maintained by the Console. For each application, the method will prepare the database packages which are to be run dynamically for the given user by checking a table which contains this information. Once the Database Synchronization Object has completed preparation of database packages, the Synchronization Server is able to look in the directory assigned to the user for any database transformation files and prepare them for download to the device.

An image of each database required for a user will be kept in the user's assigned folder in the **Users** directory. These images will be Microsoft Access™ databases. Inside a user's assigned folder databases for different applications will be kept distinct by the name of the database file.

Properties

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: Information Server Agent (abinfosvr.dll)

Date Last Revised:

Authors:

Purpose:

Compares versions of components and applications installed on device with versions assigned in Console repository. Retrieves CAB files to be downloaded to device and creates application deletion XML file for obsolete applications.

Properties

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: Login Component (ablogin.dll)

Date Last Revised:

Authors:

Purpose: The Login Component is used as an interface between the system login application and the repository for user information.

The Login Component is used as an interface between the system login application and the repository for user information.

Properties

Name: Data Type:

Name: Data Type:

Name: Data Type:

Methods:

Name: CheckLogin(*Username*, *Password*) Return Type: BOOL

Parameters:

Validates username and password

Title: Login Web Application

Date Last Revised:

Authors:

Purpose:

Properties:

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: **Packager Component (abpackager.dll)**

Date Last Revised:

Authors:

Purpose: The Packager will provide a method which allows a caller to set the URL of the main web

server. When creating a CAB file for the Default Applications, this URL will be included in the CAB file and will be set in the registry when the CAB file is executed.

Properties

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: SOAP Sync Agent (absoapsync.dll)

Date Last Revised:

Authors:

Purpose:

Handles asynchronous post files transmitted to server during synchronization process. Fires an event when all asynchronous post files have been processed. Since asynchronous post files can come from several different applications, success or failure of entire process is not inherited from any one call executed by the object.

Properties

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: SOAP Server Client (abSOAPClient.dll)

Date Last Revised:

Authors:

Purpose: Executes SOAP call on the central server. Utilized by the abSOAPSvr to handle each individual asynchronous post call. Fires and event when the call returns.

Executes SOAP call on the central server. Utilized by the abSOAPSvr to handle each individual asynchronous post call. Fires and event when the call returns.

Properties:

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: **Status Server Agent (abstatussvr.dll)**

Date Last Revised:

Authors:

Purpose: Save parameters specifying current status of device to the Console data repository. This information is used by the Console to display the most recent monitoring information reported by the device.

Properties: Name: Data Type:

Name:

Data Type:

Methods: Name: Return Type:

Parameters:

Title: Synchronization Server (abSyncSvr.dll)

Date Last Revised:

Authors:

Purpose:

The Synchronization Server is a COM+ component that is invoked by the client synchronization application (*absync.asp*) to perform application version control, database synchronization, and XOOM component version control.

For the second half of initial installations, the Synchronization Server provides a method which is passed the manufacturer and model of the device and the user operating the device. The Synchronization Server uses this information to select the device dependent XOOM components for the device, the applications assigned to the user, and the database setup files for the user. The Synchronization Server packages these files and returns them to the calling device.

The Synchronization Server manages all of the server tasks which must be accomplished during a synchronization operation. The Synchronization Server utilizes several auxiliary server components to accomplish these tasks which include: (i) execution of asynchronous SOAP calls, (ii) saving of monitoring information from device, (iii) version control of xoom components, (iv) version control of applications, (v) retrieval of device configuration command file, (vi) retrieval of application database transformation command file.

Properties:

N:

Data Type:

Methods:

Name: StartInitialInstall

Return Type:

Parameters:

Called by Console to prepare CAB file for device during initial installation

Name: ExecuteSynchronization

Return Type:

Parameters: Device OEM

Called by abSync.exe on device to start the synchronization process

Name: ObtainGUID

Return Type:

Parameters:

Called by abSync.exe on device in case that device needs to recover the current transaction Id/GUID

Title: **User Store Interface (IUser)**

Date Last Revised:

Authors:

Purpose:

The User Store Interface provides a means by which the Console can extract user information from a repository. The active repository is determined by a value which can be set through the Console interface. The possible values are limited by an XML document listing the supported repositories.

Properties

Name: Data Type:

Name: Data Type:

Name: Data Type:

Methods:

Name: `ValidateUser(Username, Password)` Return Type: `BOOL`

Parameters: `Username Password`

Checks active repository to see if the username and password are valid

Name: `GetUsers(uniqueID, dictionaryUserId, dictionaryname)` Return Type: `BOOL`

Parameters: `Uniqueld DictionaryUsersID DictionaryName`

Checks active repository to see if the username and password are valid

Server Directory Structure

Upon installation of the XOOM software on a web server, a directory structure will be created to facilitate operation of the XOOM server modules.

```
+-----/[installation folder]/
|
|   +-----/home/
|   |
|   |   +-----/login()/
|   |   +-----/bin()/
|   |   +-----/db/
|   |
|   +-----/utils/
|   |   +-----/bin/
|   |
|   +-----/users/
|   |   +-----/userGuid1/
|   |   |   +-----/temp/
|   |   |   +-----/inbox/
|   |   |   +-----/outbox/
|   |   +-----/userGuid2/
|   |   |   +-----/temp/
|   |   |   +-----/inbox/
|   |   |   +-----/outbox/
|   |
|   +-----/apps/
|   |   +-----/app1/
|   |   |   +-----/dbpkg/
|   |   +-----/app2/
|   |   |   +-----/dbpkg/
|   |
|   +-----/comp/
|   |   +-----/asp/
|   |   |   +-----/home/
|   |   |   +-----/monitor/
|   |   +-----/device/
|   |   |   +-----/smb2740/
|   |   |   +-----/casI700/
|   |   +-----/standard/
|   |   |   +-----/mips/
|   |   |   +-----/arm/
|   |   |   +-----/sh3/
```

*directories followed by “(/)” are configured as virtual directories on the central web server

The **home** directory contains the web applications which provide access to XOOM Services and any supporting files.

The **utilities** directory contains files that are used by server modules to accomplish miscellaneous tasks such as CAB file creation.

The **temp** directory contains a subdirectory for each user where files will be placed in preparation for download to the user's device.

The **components** directory contains two subdirectories: **processor** and **device**. The **processor** directory contains a subdirectory for each supported processor and any XOOM components which are dependent on the target processor. These directories also contain the web server files corresponding to each supported processor. The **device** subdirectory contains a folder for each supported device and any XOOM components which must be compiled specifically for the corresponding device such as the "abdevio.dll".

XML DOCUMENT SPECIFICATIONS

| | |
|---|---|
| Async Post XML document | Defines and asynchronous post request or response |
| Device Configuration Document (deviceconfig.xml) | Created by the admin to specify configuration settings of a device |
| Data Update Document (dataupdate.xml) | Defines the database transformations which must be performed on the device; This file is deployed to the device during synchronization |
| Device Information document (info.xml) | Specifies versions of components and applications installed on device |
| Device Status document (status.xml) | Stores parameters which are used to monitor the status of a device. |
| Manifest Document (manifest.xml) | Supplies descriptive information about an application such as author, version etc..; Each XOOM application will be accompanied by a manifest document. There will also be a manifest document for the XOOM client components. |
| Data Package Document (abdatapackage.xml) | Output from the abDataFilter.exe which defines the database subset for the device |
| Connector Interaction Document (abconnector.xml) | |
| Printer Definition document | Represents a printer label |
| Printer Definition document | Stores printer settings and command definitions |
| User Store Interface Document (abusers.xml) | Defines supported system user stores. |
| Xoom Registry Info Document (xoomreginfo.xml) | Stores device and system configuration info on a device |

Title: Async Post XML document

Purpose:

Expresses and asynchronous SOAP call to be executed from the central server or the response from such a call.

Document Type Definition:

```
<!ELEMENT async_post (request, response)>
<!ELEMENT request (id, app_name, async_id, url, payload)>
<!ELEMENT response (id, app_name, async_id, payload)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT app_name (#PCDATA)>
<!ELEMENT async_id (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT payload (#PCDATA)>
```

Example:

```
<?xml version="1.0"?>
<!DOCTYPE async_post SYSTEM "async_post.dtd">
<async_post>
  <request>
    <id>7</id>
    <app_name>App1</app_name>
    <async_id>1122334455</async_id>
    <url>http://url:8080/connector</url>
    <payload>&lt;tag>Request</tag></payload>
  </request>
</async_post>
```

Title: Connector Interaction Document (abconnector.xml)

Date Last Revised:

Authors:

Purpose:

Defines parameters for an interaction as used by XOOM Connector.

Properties

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Methods:

Name:

Return Type:

Parameters:

Title: **Data Package Document (abdatapackage.xml)**

Date Last Revised:

Authors:

Purpose: The purpose of this study was to determine the effect of a 12-week, low-intensity, supervised walking program on the physical and psychological health of older adults with mild cognitive impairment (MCI).

Output from the abDataFilter.exe which defines the database subset for the device

Properties

Name:

Data Type:

Name:

Data Type:

Name:

Data Type:

Methods: A total of 100 patients with a confirmed diagnosis of COVID-19 were recruited from the intensive care unit of a tertiary care hospital. The patients were divided into two groups based on the severity of their illness: mild and severe. The mild group consisted of 50 patients, and the severe group consisted of 50 patients. The patients were monitored for 14 days, and their clinical course was recorded. The primary outcome was the time to clinical improvement, defined as the time from the onset of symptoms to the resolution of all symptoms. The secondary outcome was the time to hospital discharge. The data were analyzed using descriptive statistics and Kaplan-Meier survival analysis.

Name:

Return Type:

Parameters:

Title: Data Update Document (dataupdate.xml)

Purpose:

Defines the database transformations which must be performed on the device; This file is deployed to the device during synchronization

Document Type Definition:

```
<!ELEMENT table (fields, rowset, deletedRowSet?)>
<!ATTLIST table name CDATA "">
<!ELEMENT fields (field+)>
<!ELEMENT field EMPTY>
<!ATTLIST field
    name CDATA ""
    number CDATA "0"
    type CDATA ""
    maxLength CDATA "0"
    mayBeNull CDATA "false">
<!ELEMENT rowSet (row*)>
<!ELEMENT row (fieldData*)>
<!ELEMENT fieldData (#PCDATA)>
<!ATTLIST fieldData number CDATA "0">
<!ELEMENT deletedRowSet (deletedRow*)>
<!ELEMENT deletedRow (fieldData*)>
```

Example:

```
<?xml version="1.0" ?>
<table name="Table1">
  <fields>
    <field name="EmployeeName" number="1" type="string" maxLength="40"
mayBeNull="false"/>
    <field name="EmployeeID" number="2" type="int" mayBeNull="false"/>
    <field name="Salary" number="3" type="float" mayBeNull="false"/>
  </fields>
  <rowSet>
    <row>
      <fieldData number="1">Jane Doe</fieldData>
      <fieldData number="2">1</fieldData>
      <fieldData number="3">80000.00</fieldData>
    </row>
    <row>
```

```
<fieldData number="1">Davie Doe</fieldData>
<fieldData number="2">100</fieldData>
<fieldData number="3">125000.00" </fieldData>
</row>
<row>
  <fieldData number="1">Joe Doe</fieldData>
  <fieldData number="2">3</fieldData>
  <fieldData number="3">175000.00</fieldData>
</row>
</rowSet>
<deletedRowSet>
  <deletedRow>
    <fieldData number="2">1</fieldData>
  </deletedRow>
</deletedRowSet>
</table>
```


Title: **Device Configuration Document (deviceconfig.xml)**

Purpose: This document is used to specify configuration settings of a device.

Created by the Console admin to specify configuration settings of a device

Document Type Definition: This document is used to specify configuration settings of a device.

```
<!ELEMENT device_config (auto_run?, password?, device_id?, backlight?, memory?,
battery?, regional_settings?, clock_settings?, owner_info?, ras_config?)>
<!ELEMENT auto_run EMPTY>
<!ATTLIST auto_run enabled (true | false) "true">
<!ELEMENT password (#PCDATA)>
<!ELEMENT device_id (#PCDATA)>
<!ELEMENT backlight (brightness?, on_battery_power?, on_external_power?)>
<!ELEMENT brightness (#PCDATA)>
<!ELEMENT on_battery_power (timeout, auto_on?)>
<!ELEMENT on_external_power (timeout, auto_on?)>
<!ELEMENT timeout (#PCDATA)>
<!ELEMENT auto_on EMPTY>
<!ATTLIST auto_on enabled (true | false) "true">
<!ELEMENT memory (#PCDATA)>
<!ATTLIST memory type (storage | program) "storage">
<!ATTLIST memory units (mb | kb | bytes) "mb">
<!ELEMENT battery (on_battery_power?, on_external_power?)>
<!ELEMENT regional_settings (language?, number_format?, currency_format?, time_format?,
date_format?)>
<!ELEMENT language (#PCDATA)>
<!ELEMENT number_format (decimal_symbol?, decimal_places?, digit_grouping_symbol?,
digits_in_group?, list_separators?, negative_sign?, negative_number_format?, leading_zero?,
measurement_system?)>
<!ELEMENT decimal_symbol (#PCDATA)>
<!ELEMENT decimal_places (#PCDATA)>
<!ELEMENT digit_grouping_symbol (#PCDATA)>
<!ELEMENT digits_in_group (#PCDATA)>
<!ELEMENT list_separators (#PCDATA)>
<!ELEMENT negative_sign (#PCDATA)>
```

```

<!ELEMENT negative_number_format (#PCDATA)>
<!ELEMENT leading_zero EMPTY>
<!ATTLIST leading_zero enabled (true | false) "true">
<!ELEMENT measurement_system (#PCDATA)>
<!ELEMENT currency_format (currency_symbol?, currency_symbol_position?,
decimal_symbol?, decimal_places?, digit_grouping_symbol?, digits_in_group?,
negative_number_format?)>
<!ELEMENT currency_symbol (#PCDATA)>
<!ELEMENT currency_symbol_position (#PCDATA)>
<!ELEMENT time_format (time_style?, time_separator?, am_symbol?, pm_symbol?)>
<!ELEMENT time_style (#PCDATA)>
<!ELEMENT time_separator (#PCDATA)>
<!ELEMENT am_symbol (#PCDATA)>
<!ELEMENT pm_symbol (#PCDATA)>
<!ELEMENT date_format (short_date_format?, date_separator?, long_date_format?)>
<!ELEMENT short_date_format (#PCDATA)>
<!ELEMENT date_separator (#PCDATA)>
<!ELEMENT long_date_format (#PCDATA)>
<!ELEMENT clock_settings (home?, visiting?)>
<!ELEMENT home (time?, date?)>
<!ELEMENT visiting (time?, date?)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT owner_info (owner_name?, owner_company?, owner_email?, notes?)>
<!ELEMENT owner_name (#PCDATA)>
<!ELEMENT owner_company (#PCDATA)>
<!ELEMENT owner_email (#PCDATA)>
<!ELEMENT notes (note+)>
<!ELEMENT note (#PCDATA)>
<!ELEMENT ras_config (connection?, advanced?)>
<!ELEMENT connection (connection_name, modem, baud_rate, country_code, area_code,
phone_number, dial_timeout, dial_tone_wait, card_tone_wait, modem_commands)>
<!ELEMENT connection_name (#PCDATA)>
<!ELEMENT modem (#PCDATA)>
<!ELEMENT baud_rate (#PCDATA)>

```

```

<!ELEMENT country_code (#PCDATA)>
<!ELEMENT area_code (#PCDATA)>
<!ELEMENT phone_number (#PCDATA)>
<!ELEMENT dial_timeout (#PCDATA)>
<!ELEMENT dial_tone_wait EMPTY>
<!ATTLIST dial_tone_wait enabled (true | false) "true">
<!ELEMENT card_tone_wait (#PCDATA)>
<!ELEMENT modem_commands (#PCDATA)>
<!ELEMENT advanced (port_settings?, tcpip?, name_server?)>
<!ELEMENT port_settings (data_bits, parity, stop_bits, flow_control)>
<!ELEMENT data_bits (#PCDATA)>
<!ELEMENT parity (#PCDATA)>
<!ELEMENT stop_bits (#PCDATA)>
<!ELEMENT flow_control (#PCDATA)>
<!ELEMENT tcpip (ip_address_assignment?, ip_address?, slip?, software_compression?,
ip_header_compression?)>
<!ELEMENT ip_address_assignment (#PCDATA)>
<!ELEMENT ip_address (#PCDATA)>
<!ELEMENT slip EMPTY>
<!ATTLIST slip enabled (true | false) "true">
<!ELEMENT software_compression EMPTY>
<!ATTLIST software_compression enabled (true | false) "true">
<!ELEMENT ip_header_compression EMPTY>
<!ATTLIST ip_header_compression enabled (true | false) "true">
<!ELEMENT name_server (name_server_assignment, dns_address, secondary_dns_address,
wins_address, secondary_wins_address)>
<!ELEMENT name_server_assignment (fixed | dynamic)>
<!ELEMENT dns_address (#PCDATA)>
<!ELEMENT secondary_dns_address (#PCDATA)>
<!ELEMENT wins_address (#PCDATA)>
<!ELEMENT secondary_wins_address (#PCDATA)>

```

Title: Device Information document (info.xml)

Purpose:

Supplies versions of applications and components installed on a device. The information in this document is utilized during version control.

Document Type Definition:

```
<!ELEMENT info (processor, model, components, manifests)>
```

```
<!ELEMENT processor (#PCDATA)>
```

```
<!ELEMENT model (#PCDATA)>
```

```
<!ELEMENT components (component)>
```

```
<!ELEMENT component (id+, name+, version+)>
```

```
<!ELEMENT manifests (manifest)>
```

```
<!ELEMENT manifest (application+, packages?)>
```

```
<!ELEMENT application (name+, description+, version+)>
```

```
<!ELEMENT id (#PCDATA)>
```

```
<!ELEMENT name (#PCDATA)>
```

```
<!ELEMENT description (#PCDATA)>
```

```
<!ELEMENT version (major+, minor+, release+, build+)>
```

```
<!ELEMENT major (#PCDATA)>
```

```
<!ELEMENT minor (#PCDATA)>
```

```
<!ELEMENT release (#PCDATA)>
```

```
<!ELEMENT build (#PCDATA)>
```

```
<!ELEMENT packages (package?)>
```

```
<!ELEMENT package (#PCDATA)>
```

Example:

```
<?xml version="1.0"?>
```

```
<!DOCTYPE info SYSTEM "info.dtd">
```

```
<info>
```

```
  <processor>SH3</processor>
```

```
  <model>Jornada</model>
```

```
  <components>
```

```
    <component>
```

```
      <id>12345678901234567890123456789012</id>
```

```
      <name>abinfo.dll</name>
```

```
      <version>
```

```

        <major>1</major>
        <minor>1</minor>
        <release>1</release>
        <build>1</build>
    </version>
</component>
<component>
    <id>12345678901234567890123456789012</id>
    <name>abaspex.dll</name>
    <version>
        <major>2</major>
        <minor>1</minor>
        <release>34</release>
        <build>1439</build>
    </version>
</component>
</components>
<manifests><manifest>
    <application>
        <id>12345678901234567890123456789012</id>
        <name>App1</name>
        <description>Application Description</description>
        <version>
            <major>1</major>
            <minor>1</minor>
            <release>1</release>
            <build>1</build>
        </version>
    </application>
    <packages>
        <package>Package 1</package>
        <package>Package 2</package>
    </packages>
</manifest>
<manifest>
    <application>
        <id>12345678901234567890123456789012</id>
        <name>App2</name>
        <description>Application Description II</description>
        <version>
            <major>4</major>
            <minor>1</minor>
            <release>1</release>
            <build>45</build>
        </version>
    </application>
    <packages>
        <package>Package 3</package>
    </packages>
</manifest>

```

</manifests></info>

Title: **Device Status document (status.xml)**

Purpose:

Stores device parameters such as memory level and battery level which are used to monitor the operating status of the device.

Document Type Definition:

```
<!ELEMENT status (battery, memory, storage)>
<!ELEMENT memory (load, phys_total, phys_avail)>
<!ELEMENT load (#PCDATA)>
<!ELEMENT phys_total (#PCDATA)>
<!ELEMENT phys_avail (#PCDATA)>
<!ELEMENT storage (size, free)>
<!ELEMENT size (#PCDATA)>
<!ELEMENT free (#PCDATA)>
<!ELEMENT battery (ac_line_status, charge_status, life_percent, chemistry)>
<!ELEMENT ac_line_status (#PCDATA)>
<!ELEMENT charge_status (#PCDATA)>
<!ELEMENT life_percent (#PCDATA)>
<!ELEMENT chemistry (#PCDATA)>
```

Example:

```
<?xml version="1.0"?>
<!DOCTYPE status SYSTEM "status.dtd">
<status>
  <memory>
    <load>42</load>
    <phys_total>9862144</phys_total>
    <phys_avail>5744640</phys_avail>
  </memory>
  <storage>
    <size>6712320</size>
    <free>3156980</free>
  </storage>
  <battery>
    <ac_line_status>Offline</ac_line_status>
    <charge_status>Critical</charge_status>
    <life_percent>0</life_percent>
    <chemistry>Lilon</chemistry>
  </battery>
```

</status>

Title: **Manifest Document (manifest.xml)**

Purpose:

Supplies descriptive information about an application such as author, version etc.; Each XOOM application will be accompanied by a manifest document. There will also be a manifest document for the XOOM client components.

Document Type Definition:

```
<!ELEMENT manifest (application+, packages?)>
<!ELEMENT application (name+, description+, version+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT version (major+, minor+, release+, build+)>
<!ELEMENT major (#PCDATA)>
<!ELEMENT minor (#PCDATA)>
<!ELEMENT release (#PCDATA)>
<!ELEMENT build (#PCDATA)>
<!ELEMENT icon (#PCDATA)>
<!ELEMENT app-type (#PCDATA)>
<!ELEMENT app-exe (#PCDATA)>
<!ELEMENT packages (package?)>
<!ELEMENT package (#PCDATA)>
```

Example:

```
<?xml version="1.0" ?>
<manifest>
  <application>
    <name>Application 1</name>
    <description>Application Description</description>
    <version>
      <major>1</major>
      <minor>1</minor>
      <release>1</release>
      <build>1</build>
    </version>
  </application>
  <packages>
    <package>Package 1</package>
    <package>Package 2</package>
  </packages>
```

</manifest>

Title: Printer Definition document ([printrname]def.xml)

Purpose:

This document defines the command strings and settings associated with a specific make of printer. The name of the document includes the name of the manufacturer of the printer followed by the letters "d-e-f". This document is used, in conjunction with the XSL document corresponding to the printer, by the printer object in abstdio to facilitate printing using the object.

Document Type Definition:

```
<!ELEMENT printerdef (globals, textfonts, barcodefonts, graphics, auxhardware)>
<!ELEMENT globals (printescchars, labelheader, labelfooter, vertical_cmd, wakeupchars,
databuffer, baud, parity, stopbits, flowcontrol)>
<!ELEMENT printescchars (#PCDATA)>
<!ELEMENT labelheader (#PCDATA)>
<!ELEMENT labelfooter (#PCDATA)>
<!ELEMENT vertical_cmd (#PCDATA)>
<!ELEMENT wakeupchars (#PCDATA)>
<!ELEMENT databuffer (#PCDATA)>
<!ELEMENT baud (#PCDATA)>
<!ELEMENT parity (#PCDATA)>
<!ELEMENT stopbits (#PCDATA)>
<!ELEMENT flowcontrol (#PCDATA)>
<!ELEMENT textfonts (textfont+)>
<!ELEMENT textfont (name, desc, escape, commands)>
<!ELEMENT barcodefonts (barcodefont+)>
<!ELEMENT barcodefont (name, desc, escape, commands)>
<!ELEMENT graphics (graphic+)>
<!ELEMENT graphic (name, desc, escape, commands)>
<!ELEMENT auxhardware (hardware+)>
<!ELEMENT hardware (name, desc, escape, commands)>
<!ELEMENT name (#PCDATA)>
<!ATTLIST name
  baseht CDATA
  basewd CDATA>
<!ELEMENT desc (#PCDATA)>
<!ELEMENT escape (#PCDATA)>
<!ELEMENT commands (size+, command+)>
<!ELEMENT size (#PCDATA)>
<!ATTLIST size
  index CDATA
  lineht CDATA>
<!ELEMENT command (#PCDATA)>
```

```
<!ATTLIST command  
  index CDATA  
  func CDATA>
```

Example: 

Title: Printer Label document

Purpose: Represents a printer label which can be printed on a printer. These documents will be created by application programmers when printing using the `abstdio` control.

Represents a printer label which can be printed on a printer. These documents will be created by application programmers when printing using the `abstdio` control.

The label consists of two sections: a header and a body. The header section contains values needed to determine the starting position of the label and other values which are global to the label. The body section is comprised of line elements. The data enclosed in each line element is printed to the printer. The data attribute specifies the type of data to be printed. The type attribute specifies which font or barcode type is to be used. The size attribute specifies the physical size of the data printed. The len attribute is only interpreted when the type of data printed is border. This attribute specifies the physical length of the border in printer dots.

Document Type Definition: `<?xml version="1.0"?>`

```
<!ELEMENT label (header, body)>
<!ELEMENT header (copies, x_start, y_start, height, orientation)>
<!ELEMENT x_start (#PCDATA)>
<!ELEMENT y_start (#PCDATA)>
<!ELEMENT height (#PCDATA)>
<!ELEMENT orientation (#PCDATA)>
<!ELEMENT body (line)>
<!ELEMENT line (#PCDATA)>
<!ATTLIST line
  data (text | barcode | border) "text"
  type CDATA
  size CDATA
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  len CDATA>
```

Example: `<?xml version="1.0"?>`

```
<?xml version="1.0"?>
<label>
  <header> <x_start>5</x_start>
            <y_start>5</y_start>
            <orientation>horizontal</orientation>
            <height>200</height>
            <copies>200</copies> </header>
  <body> <line data="text" type="1" size="0">Hello Xoom</line>
         <line data="barcode" type="0" size="2">Code 128 test</line>
         <line data="border" type="1" size="3" len="350"/> </body>
```

Title: User Store Interface Document (abusers.xml)

Purpose:

Defines the user stores supported by the XOOM system. The Console reads this document to determine which options should be given to an administrator when selecting the user repository to be used by the system. Once a system administrator has selected a repository, the document is used to determine which component should be called to interface with the specified repository.

Document Type Definition:

```
<!ELEMENT Xoom_Users (Xoom_User+)>
```

```
<!ELEMENT Xoom_User (Desc, ProgID)>
```

```
<!ELEMENT Desc (#PCDATA)>
```

```
<!ELEMENT ProgID (#PCDATA)>
```

Example:

```
<?xml version="1.0"?>
<!DOCTYPE Xoom_Users SYSTEM "xmlUsers.dtd">
<Xoom_Users>
  <Xoom_User>
    <Desc>Xoom</Desc>
    <ProgID>abUsers_Xoom.Users</ProgID>
  </Xoom_User>
  <Xoom_User>
    <Desc>Active Directory</Desc>
    <ProgID>abUsers_ADSI.Users</ProgID>
  </Xoom_User>
  <Xoom_User>
    <Desc>R/3</Desc>
    <ProgID>abUsers_R3.Users</ProgID>
  </Xoom_User>
</Xoom_Users>
```

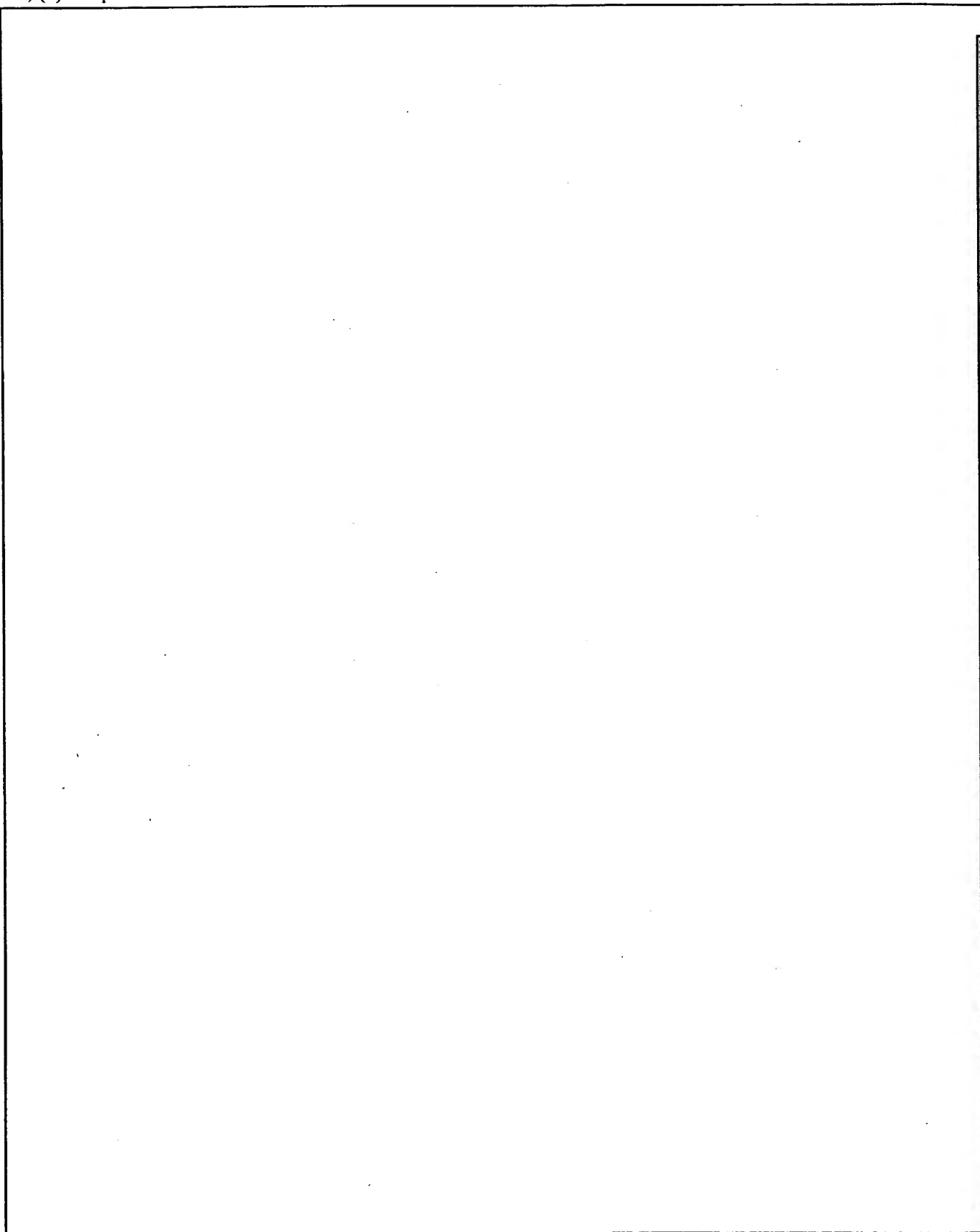
Title: Xoom Registry Info Document (xoomreginfo.xml)

Purpose: Stores xoom system information and configuration data on a device. This information is used by various device components to determine system settings and configuration variables. The "serversyncURL" specifies the path to the XOOM soap router for calling system server objects. The "serverputURL" specifies the path to the server directory where files referenced in async-post SOAP calls should be placed.

Document Type Definition:
<ELEMENT serversyncURL (#PCDATA)>
<ELEMENT serverputURL (#PCDATA)>

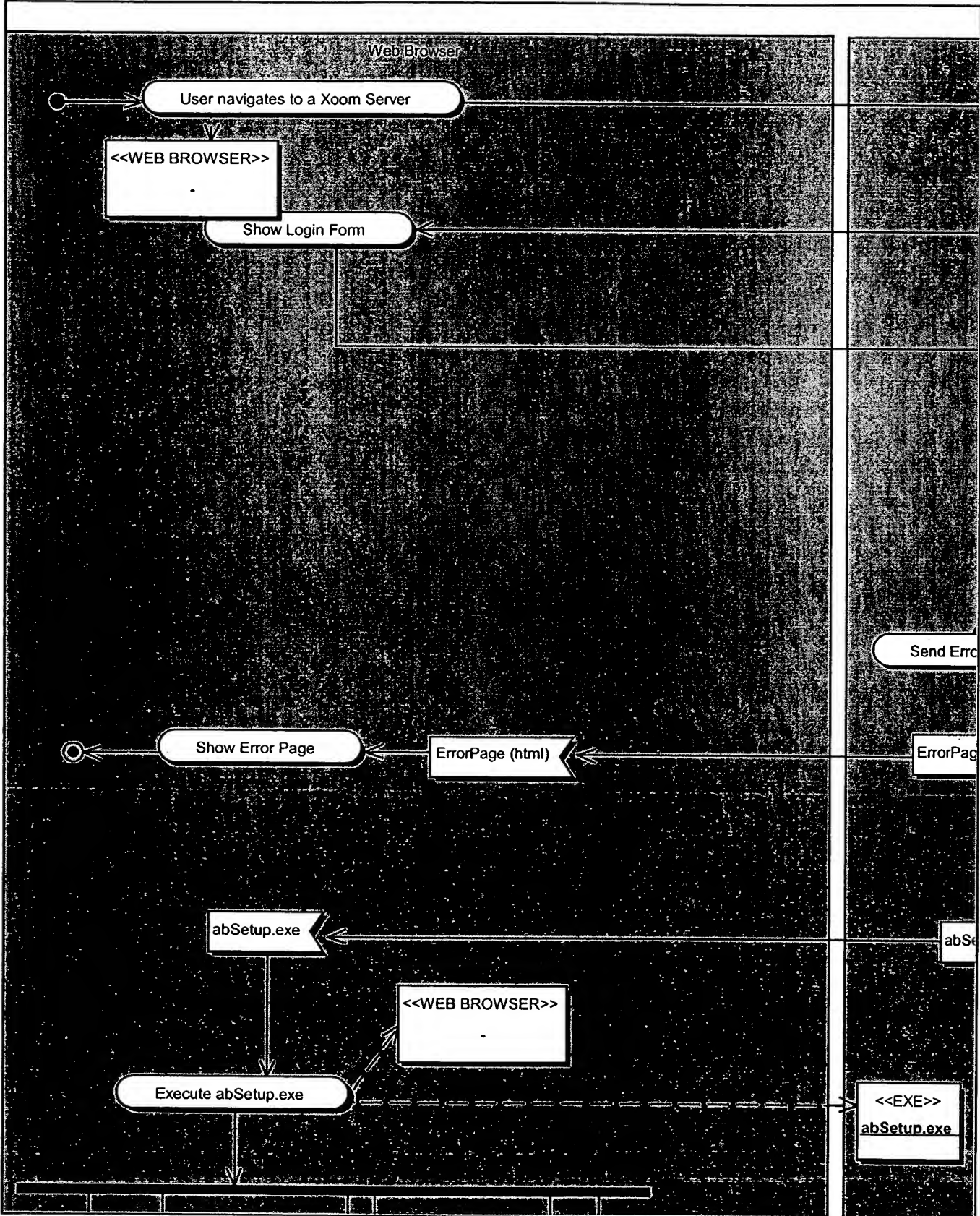
Example:

1.1, (1)Setup



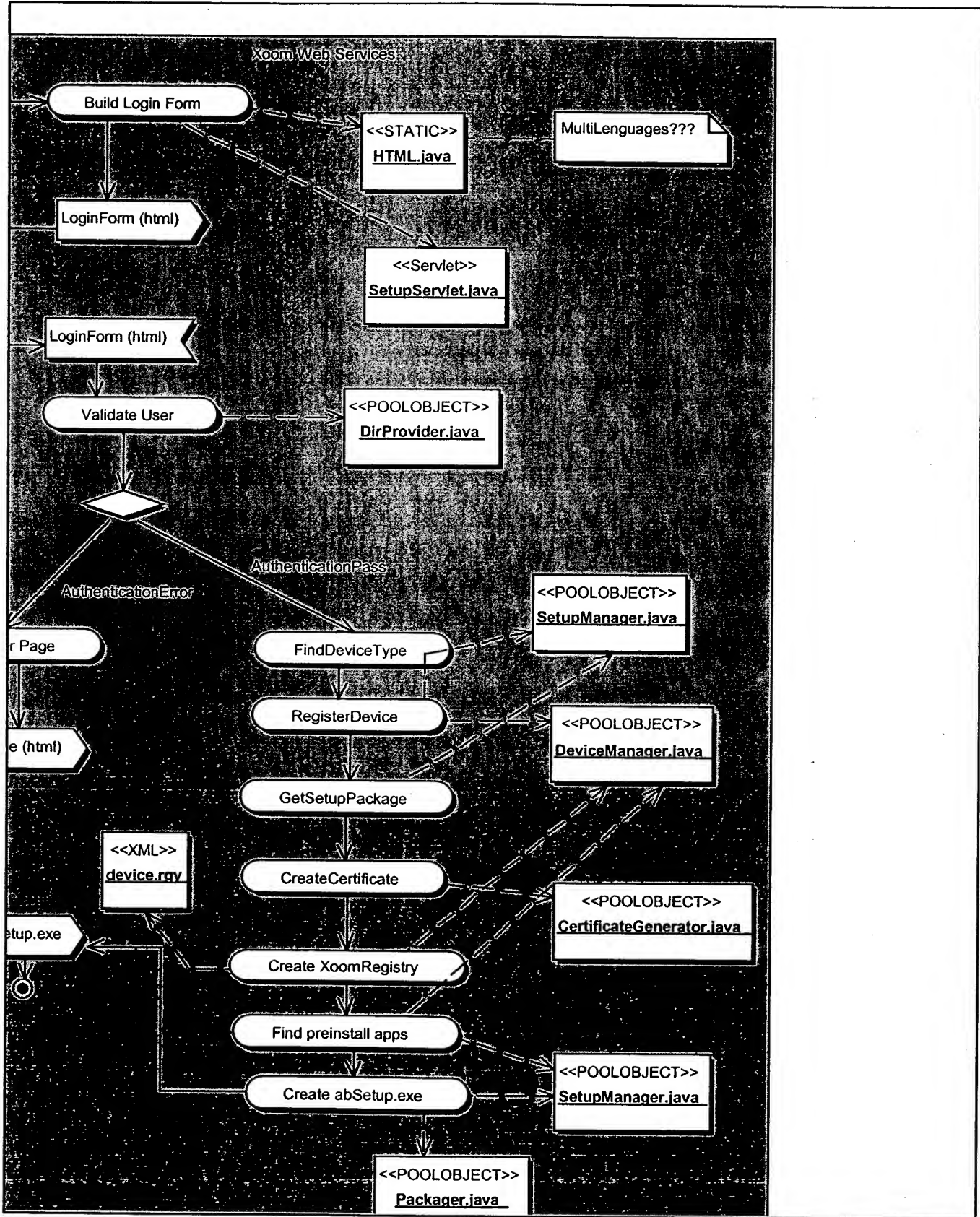
1.1, (1)Setup

1.2, (1)Setup



1.2, (1)Setup

1.3, (1)Setup

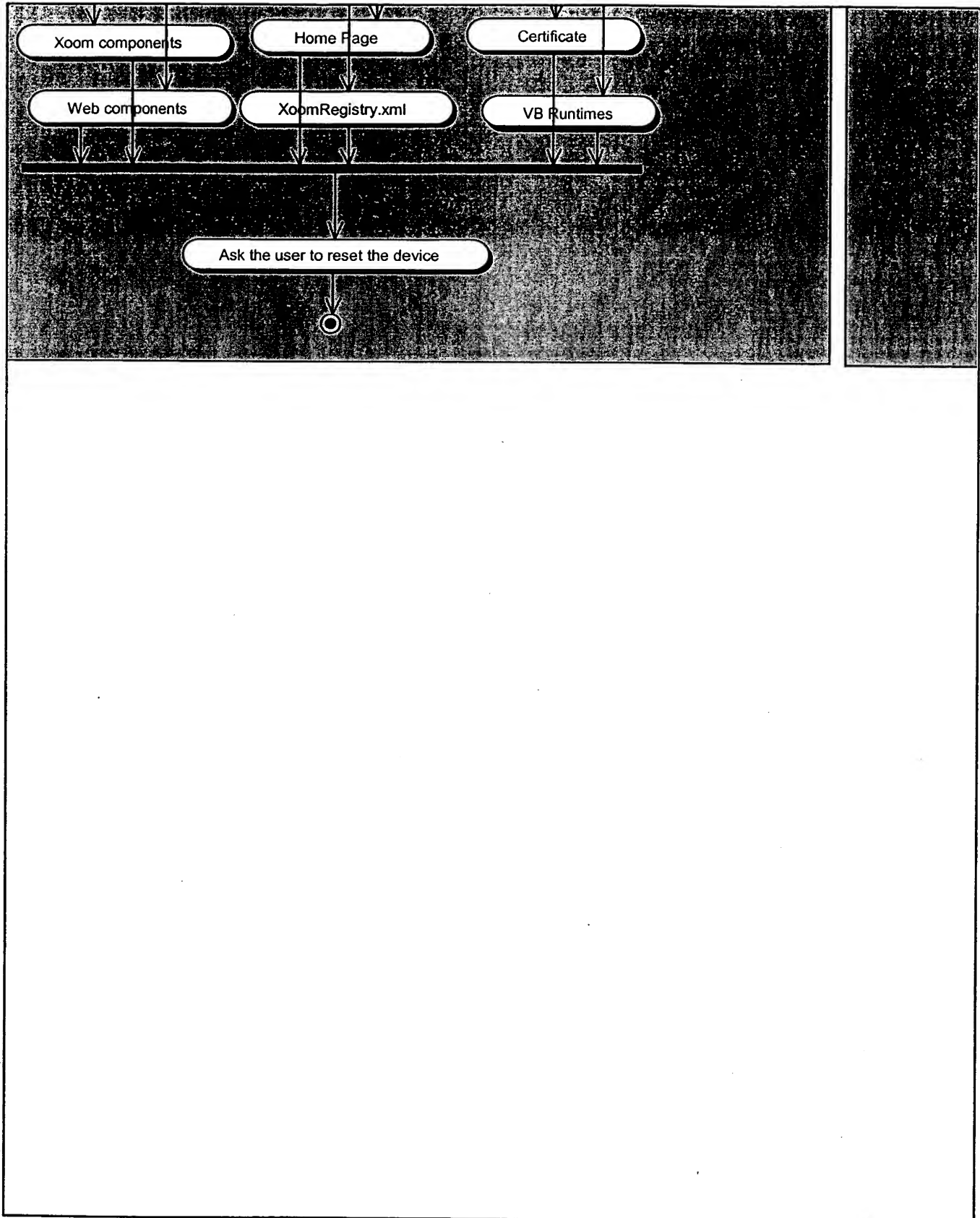


1.3, (1)Setup

2.1, (1)Setup

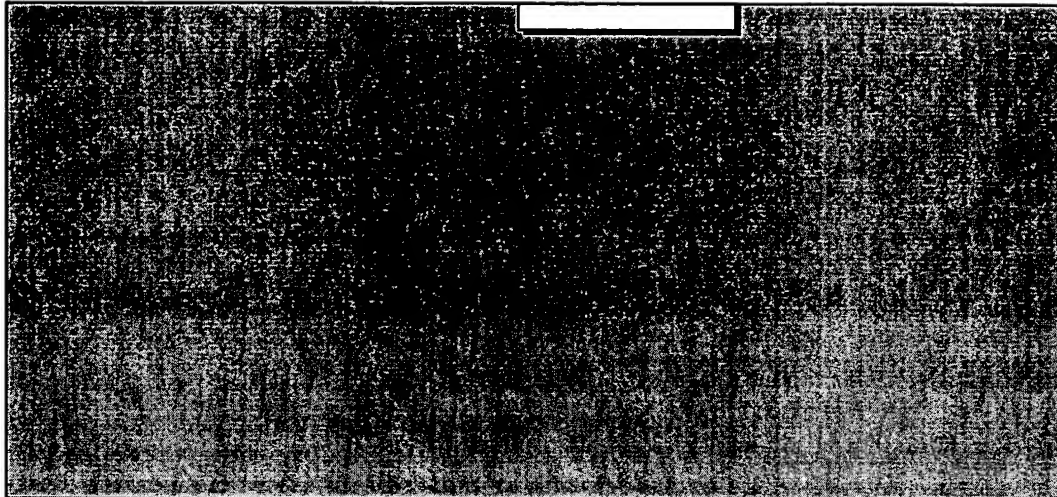
2.1, (1)Setup

2.2, (1)Setup



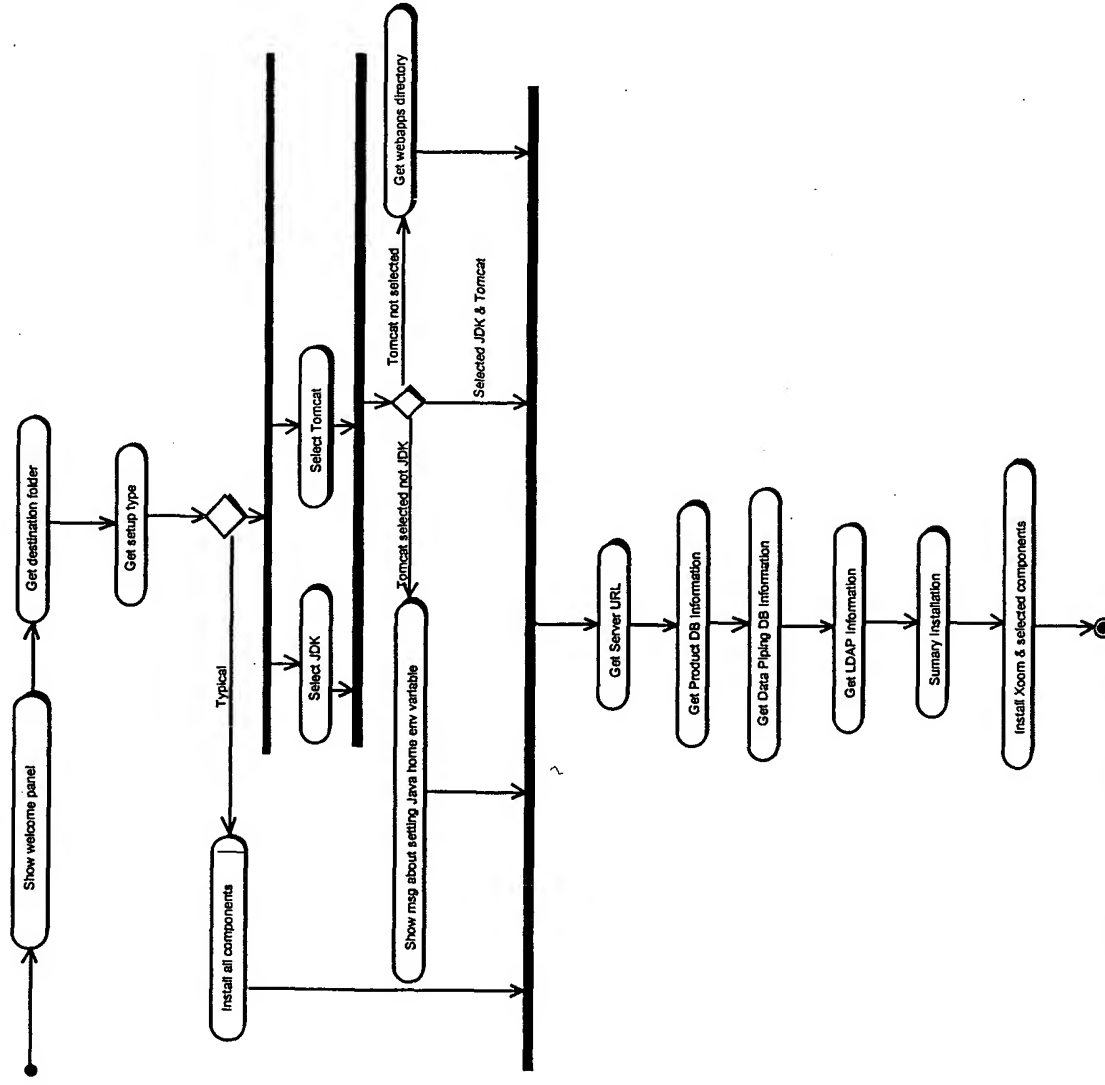
2.2, (1)Setup

2.3, (1)Setup



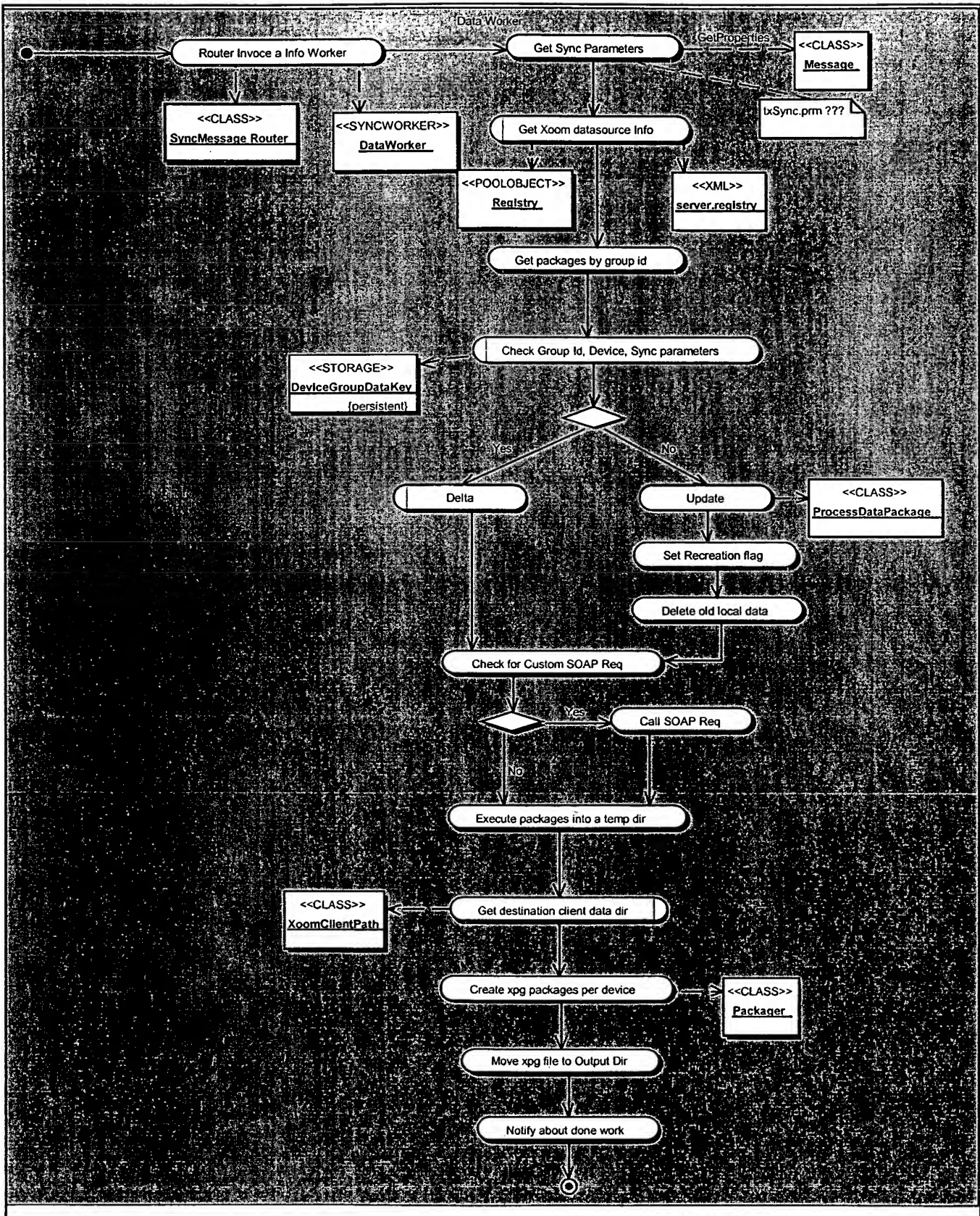
2.3, (1)Setup

1.1, (10)Xoom Installation



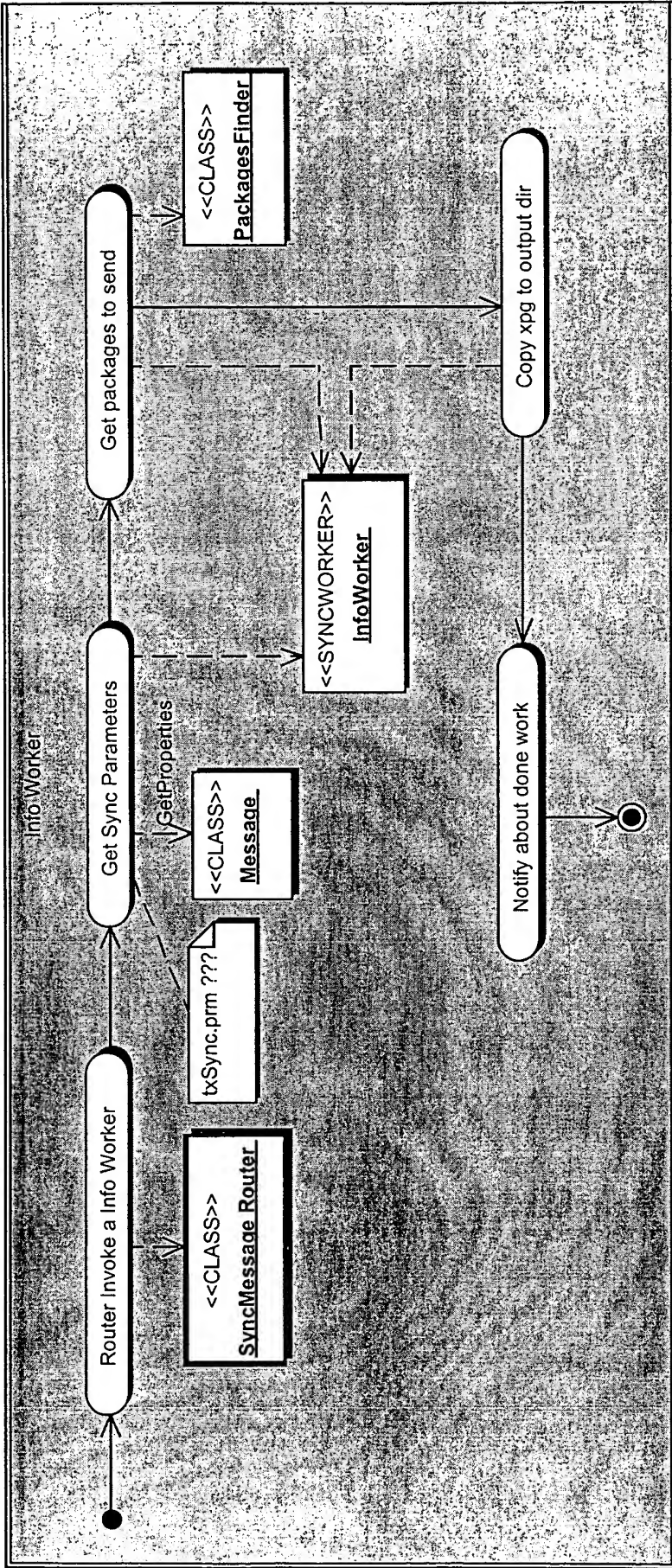
1.1, (10)Xoom Installation

1.1, (2)Data Worker



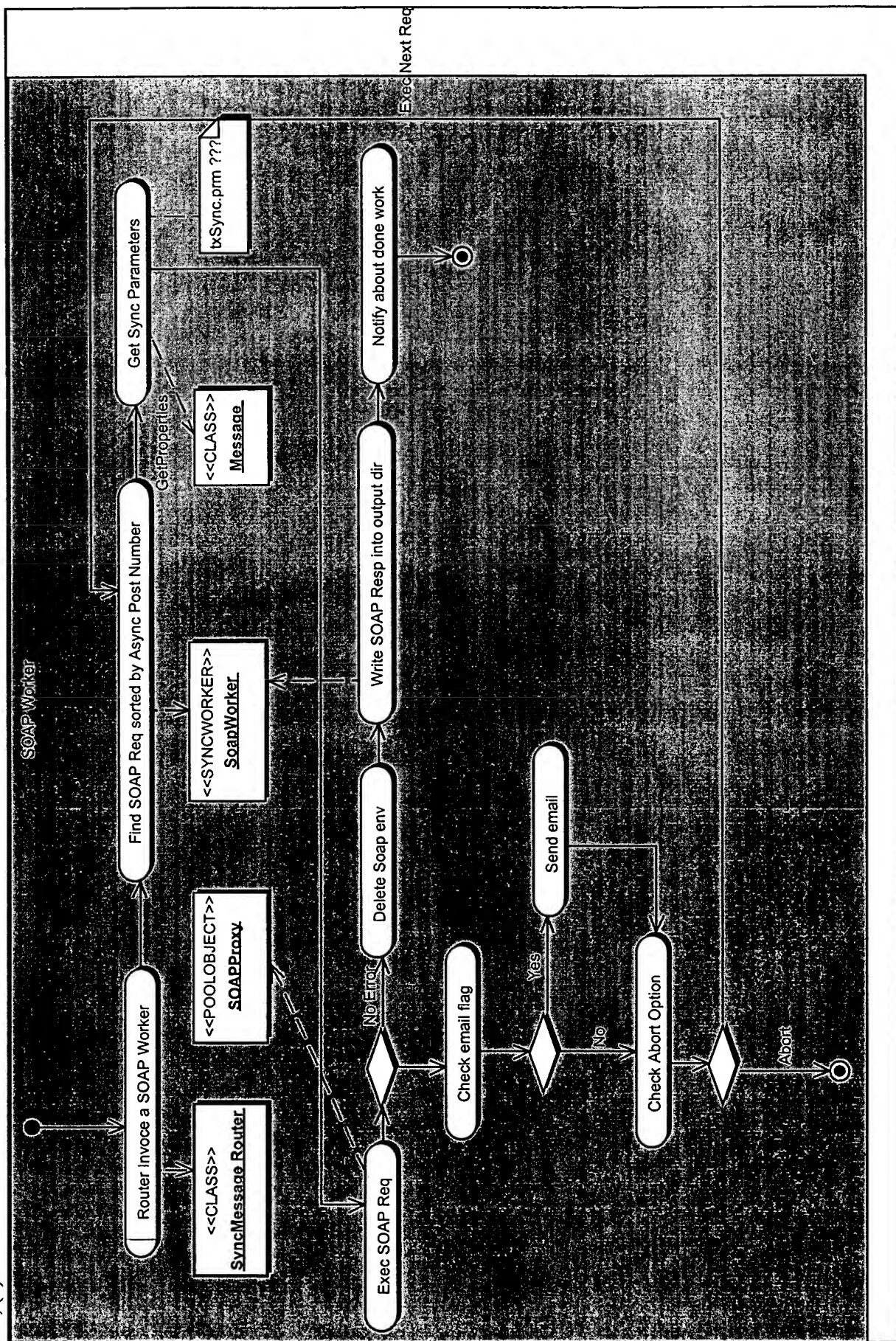
1.1, (2)Data Worker

1.1, (2)Info Worker



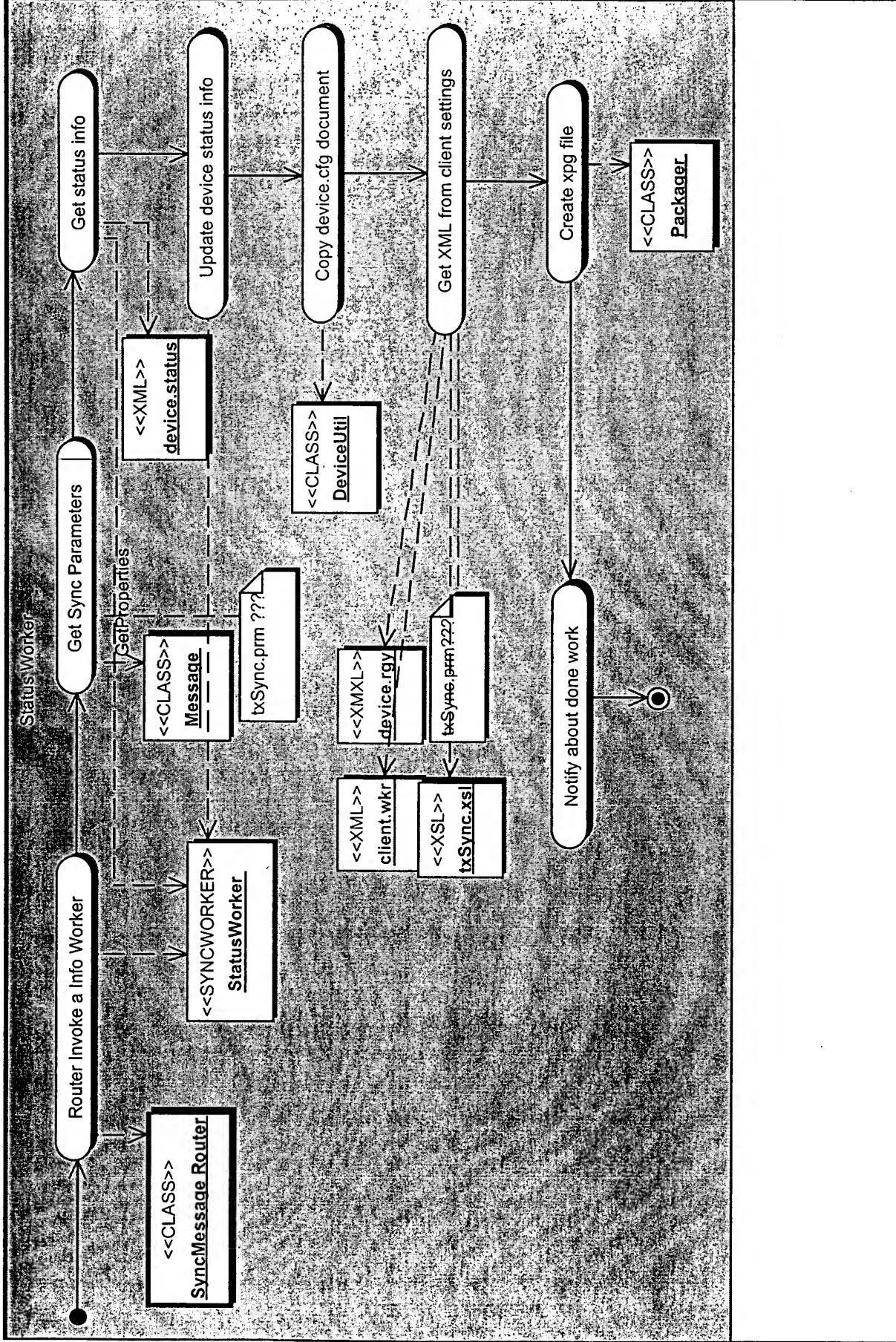
1.1. (2)Info Worker

1.1, (2)SOAP Worker



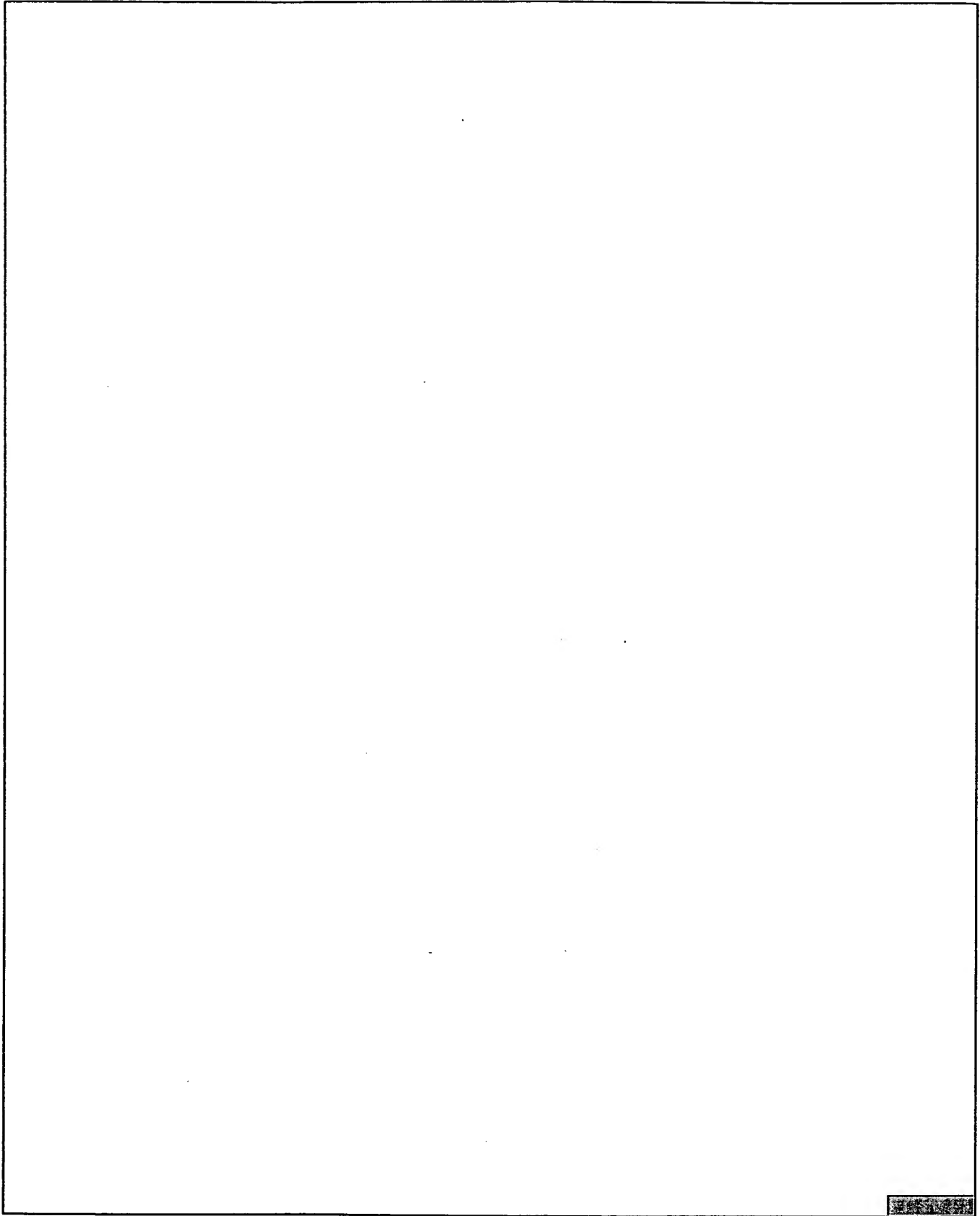
1.1, (2)SOAP Worker

1.1, (2)Status Worker



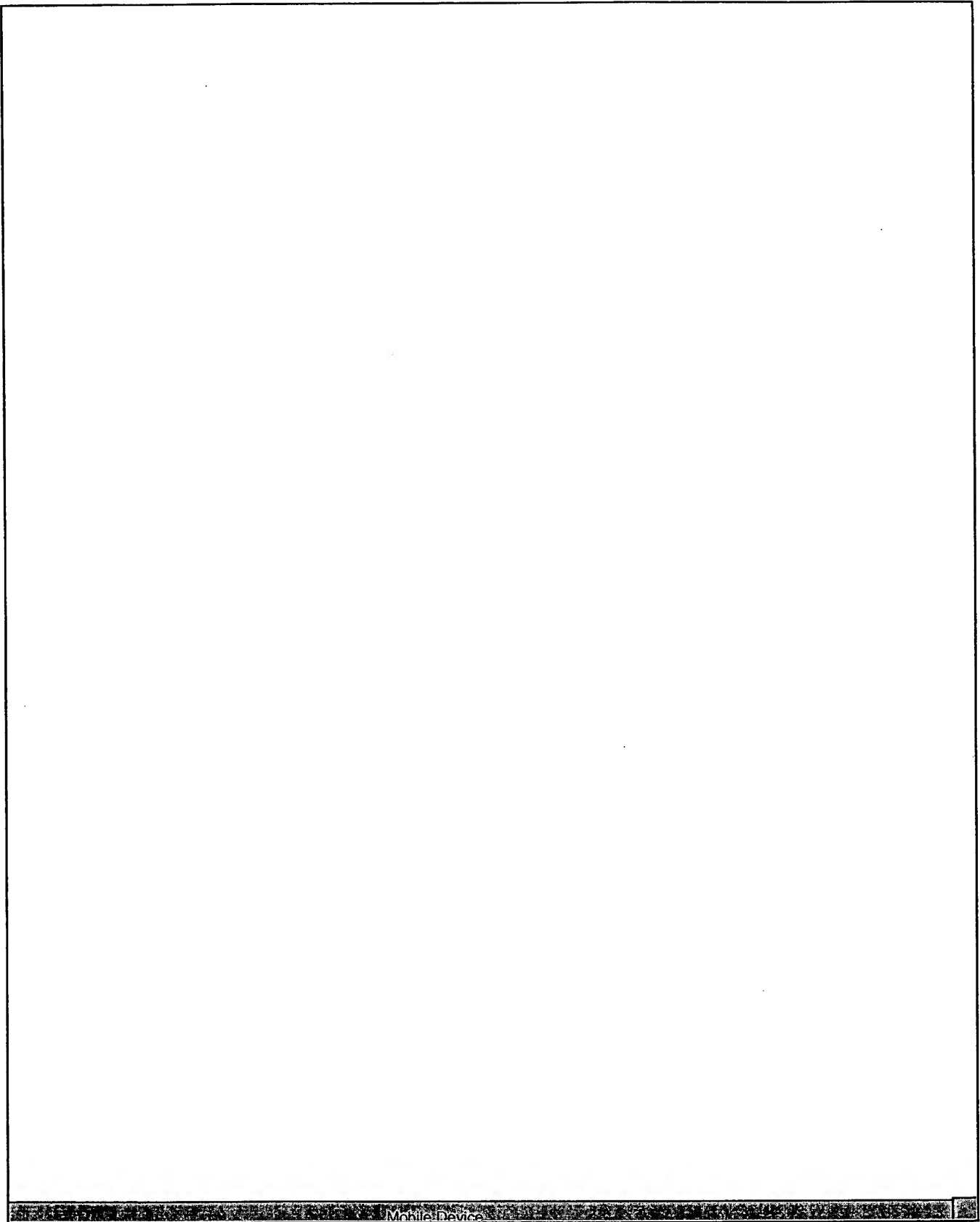
1.1, (2)Status Worker

1.1, (2)txSync



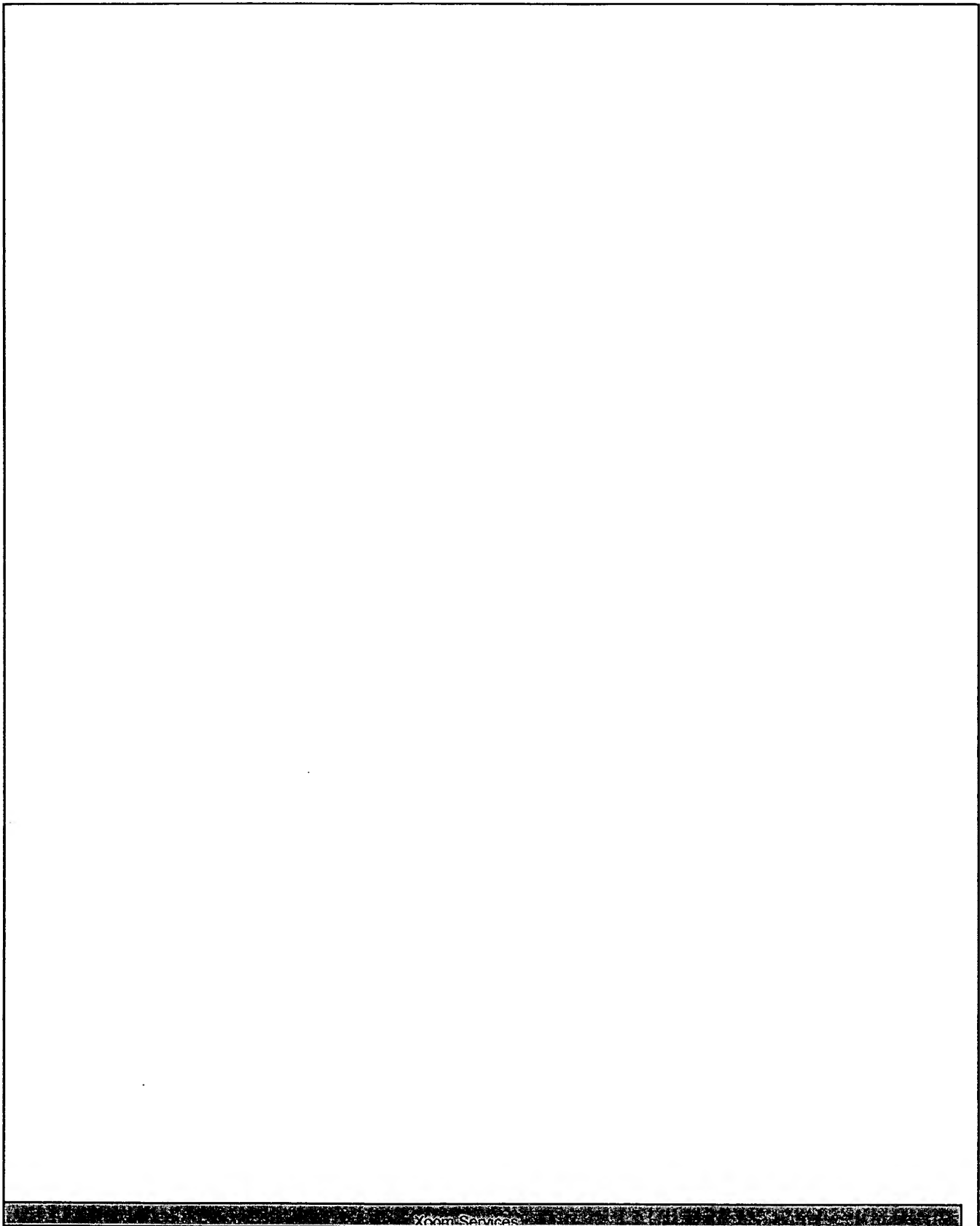
1.1, (2)txSync

1.2, (2)txSync



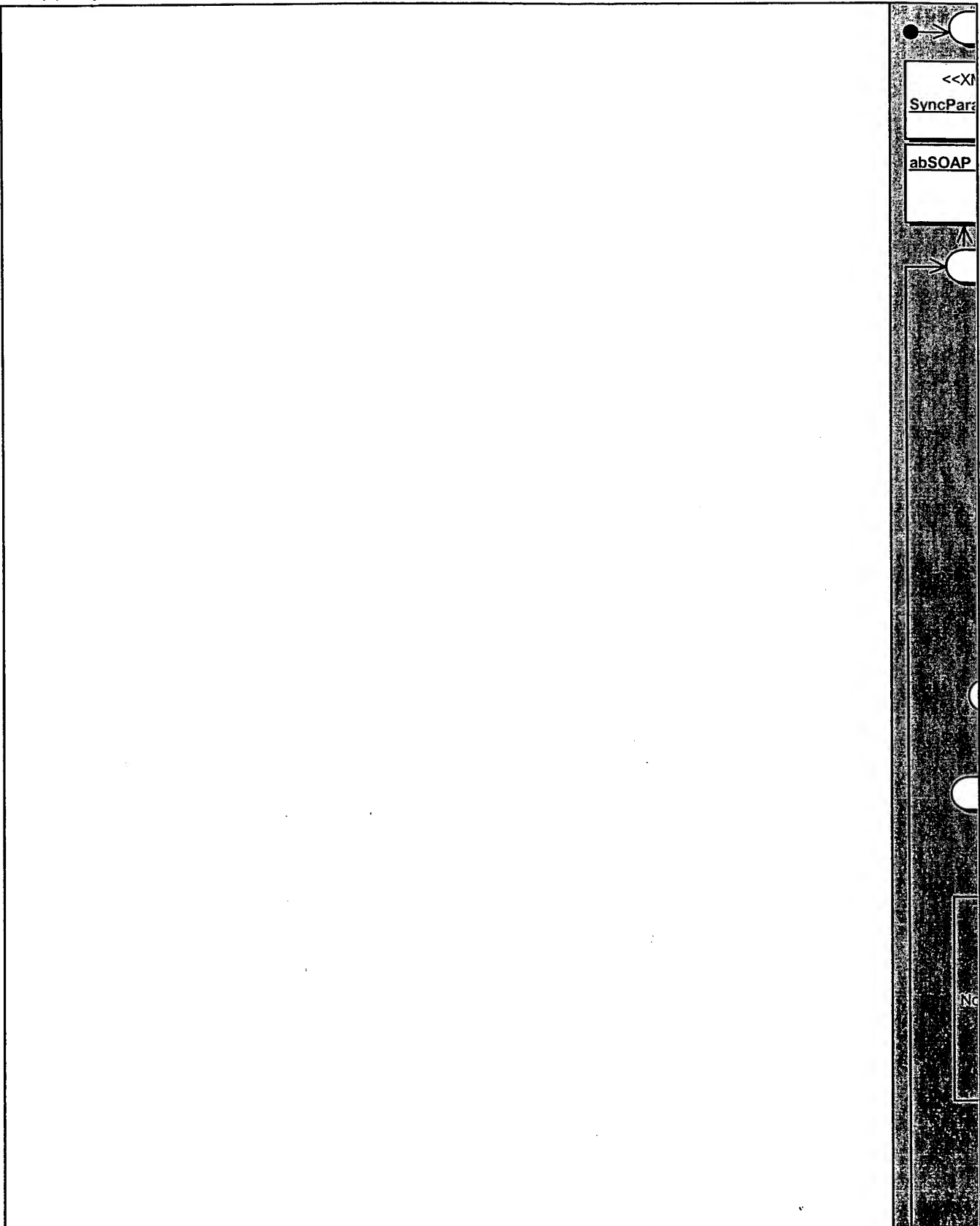
1.2, (2)txSync

1.3, (2)txSync



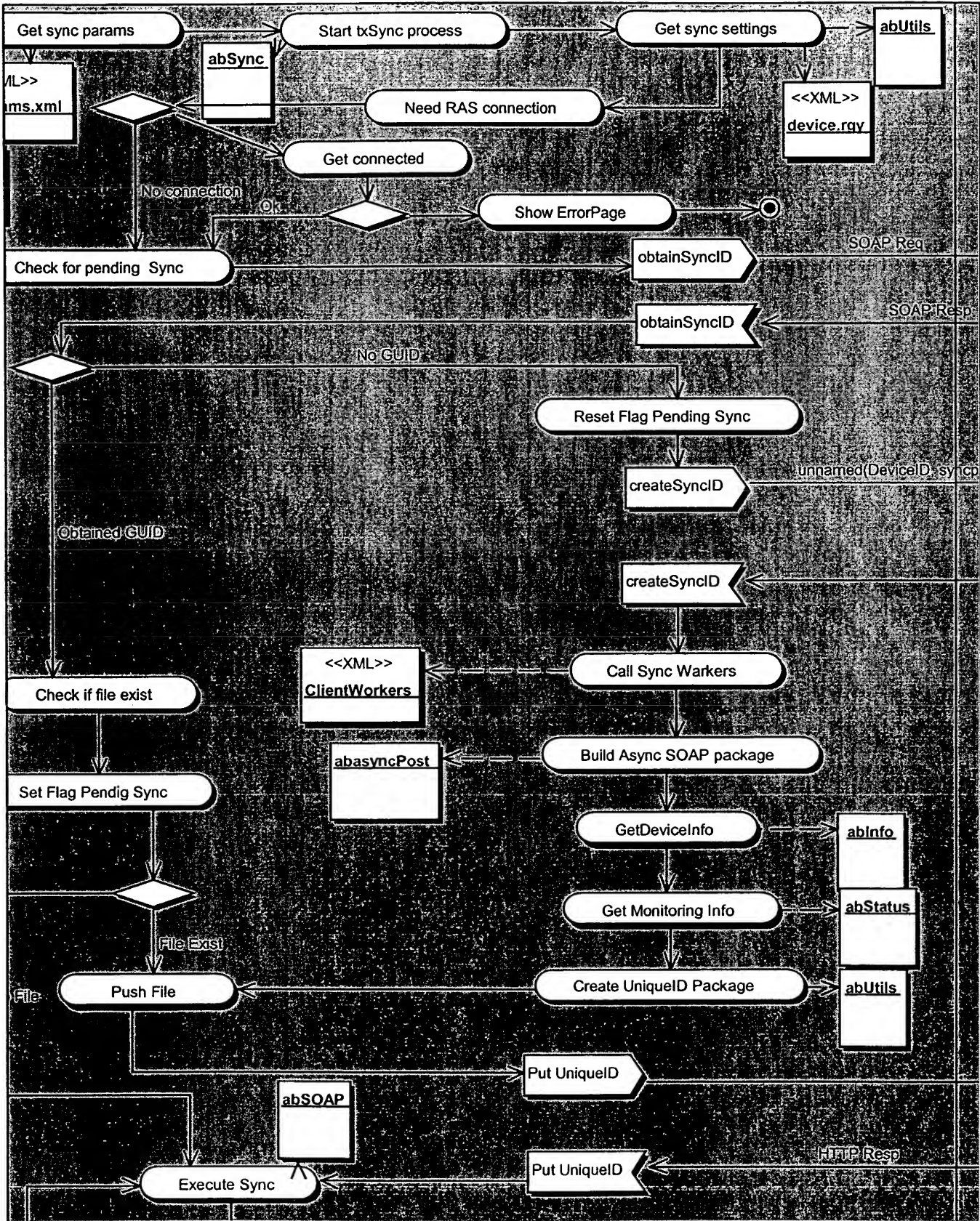
1.3, (2)txSync

2.1, (2)txSync



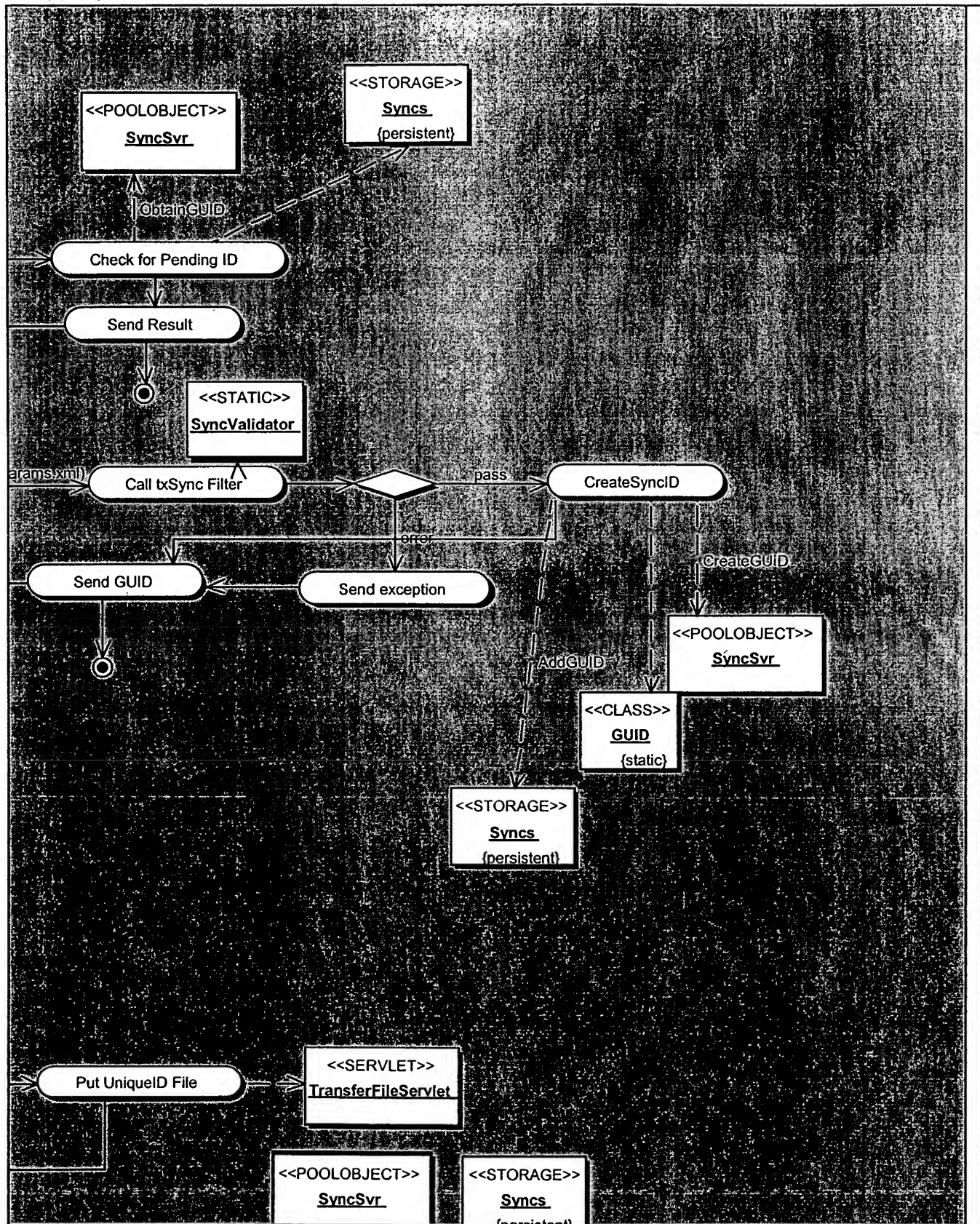
2.1, (2)txSync

2.2, (2)txSync



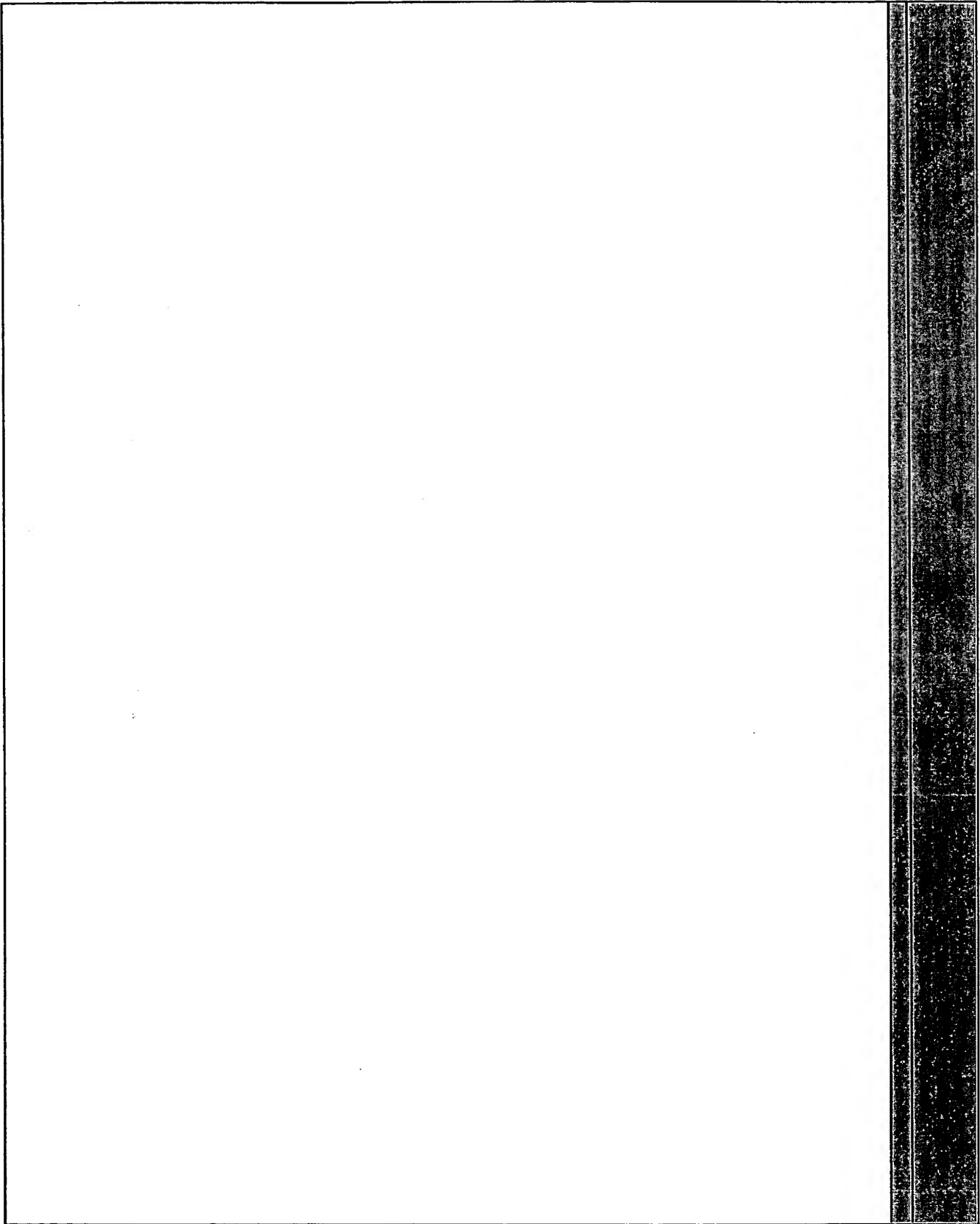
2.2, (2)txSync

2.3, (2)txSync



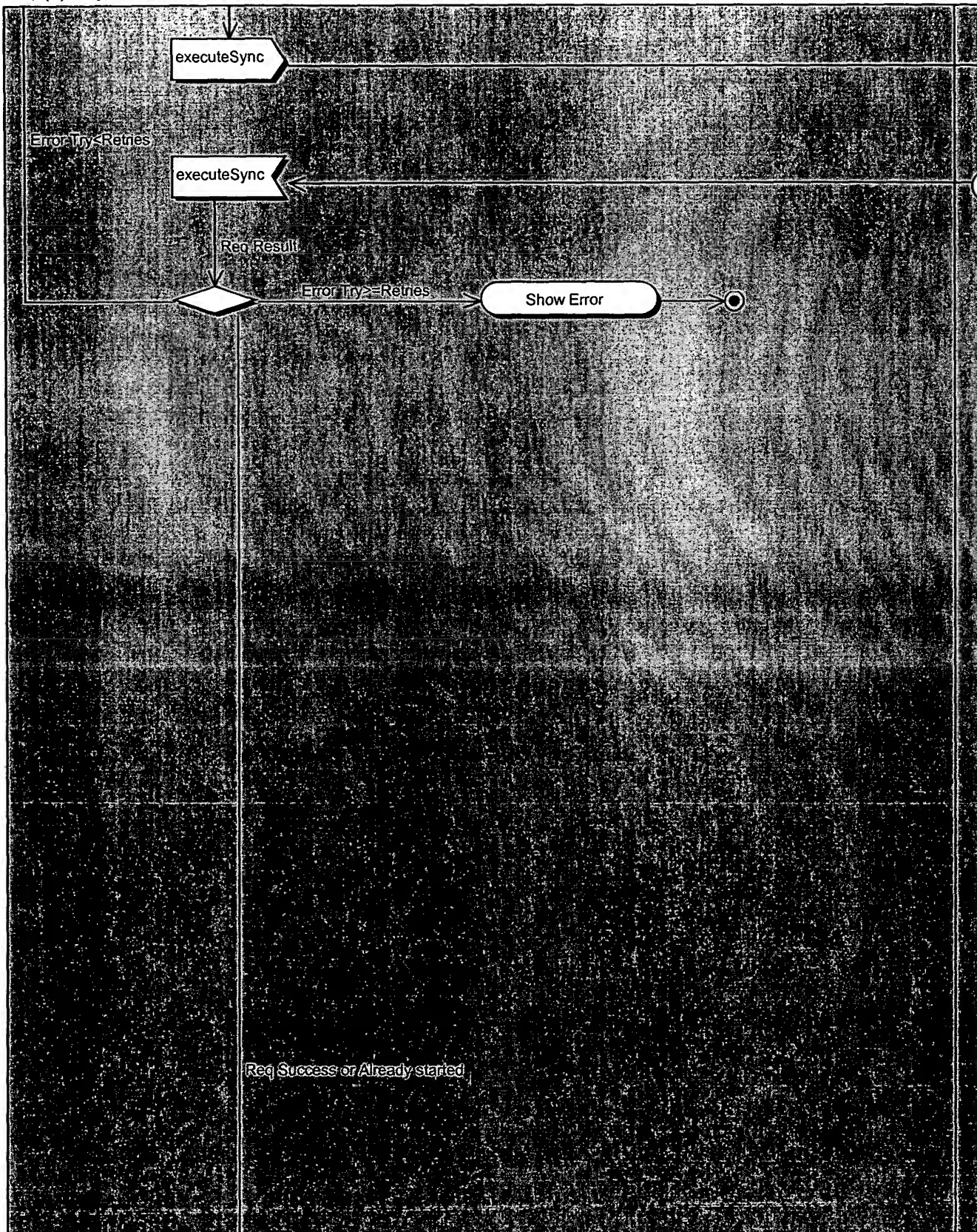
2.3, (2)txSync

3.1, (2)txSync



3.1, (2)txSync

3.2, (2)txSync



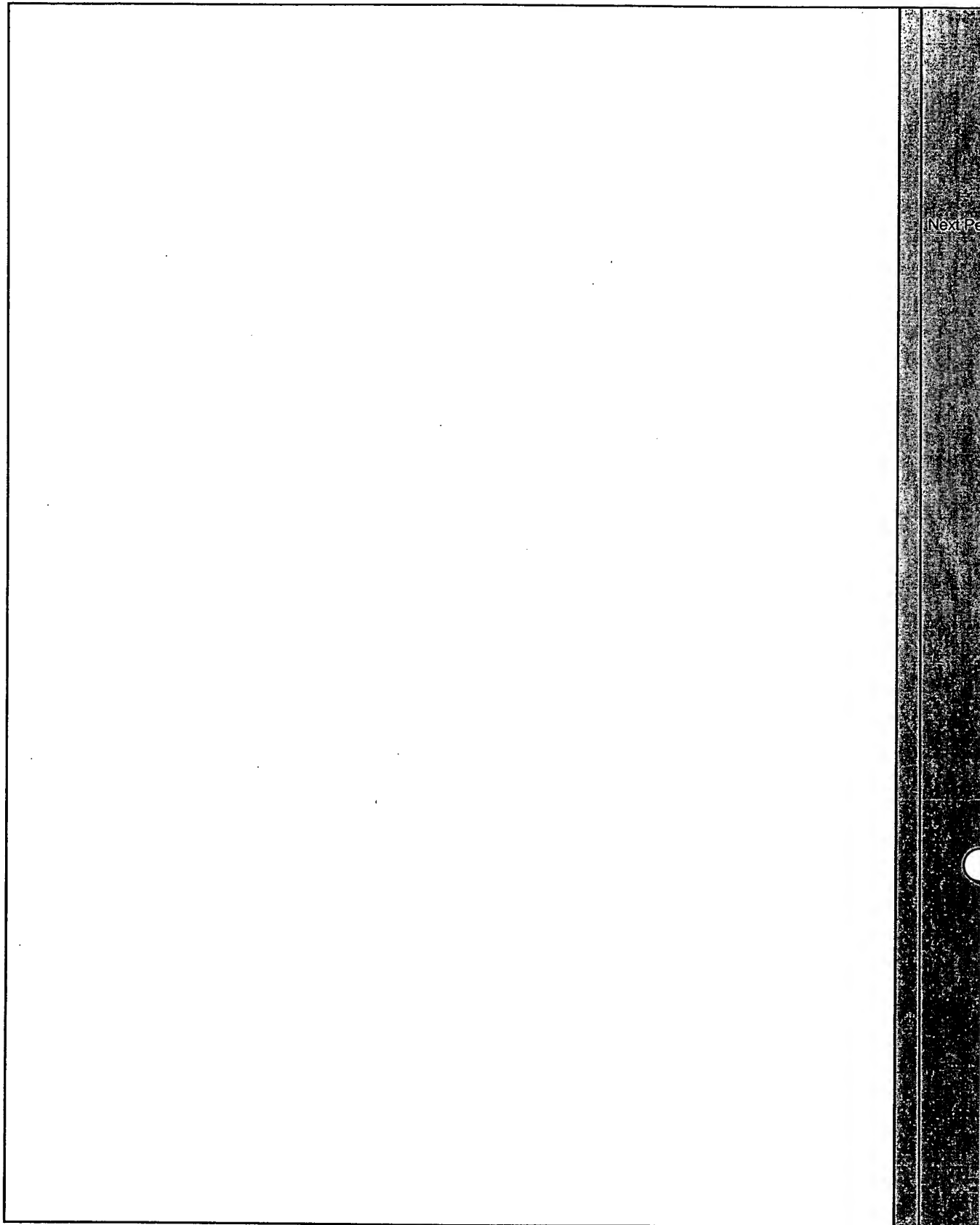
3.2, (2)txSync

```

    graph TD
        Start(( )) --> ExecuteSync[Execute Sync]
        ExecuteSync --> CheckProcess[Check if process status]
        CheckProcess --> Decision1{ }
        Decision1 -- "No form" --> SendResult[Send Exec Result]
        Decision1 -- "Failed" --> StartWorkersFailed[Start workers with failed status]
        Decision1 -- "Status new" --> GetUser[Get User Information]
        GetUser --> CheckFile[Check if file exist]
        CheckFile --> Decision2{ }
        Decision2 -- "No File" --> SendResult
        Decision2 -- "Get file" --> Uncompress[Uncompress file]
        Uncompress --> FindWorkers[Find Reg independet SynWorkers]
        FindWorkers --> CreateEntry[Create a monitoring entry]
        CreateEntry --> PublishMsg[Publish a start workers message]
        PublishMsg --> Parallel1[ ]
        Parallel1 --> MsgToSoap[MsgToSoapWorker]
        Parallel1 --> MsgToInfo[MsgToInfoWorker]
        Parallel1 --> MsgToStatus[MsgToStatusWorker]
        Parallel1 --> MsgToCustom[MsgToCustomWorker]
        Parallel1 --> Parallel2[ ]
        Parallel2 --> SetProcessing[Set Sync status to processing]
        SetProcessing --> End(( ))
        End --> End2(( ))
        End2 --> End3(( ))
        End3 --> End4(( ))
        End4 --> End5(( ))
        End5 --> End6(( ))
        End6 --> End7(( ))
        End7 --> End8(( ))
        End8 --> End9(( ))
        End9 --> End10(( ))
        End10 --> End11(( ))
        End11 --> End12(( ))
        End12 --> End13(( ))
        End13 --> End14(( ))
        End14 --> End15(( ))
        End15 --> End16(( ))
        End16 --> End17(( ))
        End17 --> End18(( ))
        End18 --> End19(( ))
        End19 --> End20(( ))
        End20 --> End21(( ))
        End21 --> End22(( ))
        End22 --> End23(( ))
        End23 --> End24(( ))
        End24 --> End25(( ))
        End25 --> End26(( ))
        End26 --> End27(( ))
        End27 --> End28(( ))
        End28 --> End29(( ))
        End29 --> End30(( ))
        End30 --> End31(( ))
        End31 --> End32(( ))
        End32 --> End33(( ))
        End33 --> End34(( ))
        End34 --> End35(( ))
        End35 --> End36(( ))
        End36 --> End37(( ))
        End37 --> End38(( ))
        End38 --> End39(( ))
        End39 --> End40(( ))
        End40 --> End41(( ))
        End41 --> End42(( ))
        End42 --> End43(( ))
        End43 --> End44(( ))
        End44 --> End45(( ))
        End45 --> End46(( ))
        End46 --> End47(( ))
        End47 --> End48(( ))
        End48 --> End49(( ))
        End49 --> End50(( ))
        End50 --> End51(( ))
        End51 --> End52(( ))
        End52 --> End53(( ))
        End53 --> End54(( ))
        End54 --> End55(( ))
        End55 --> End56(( ))
        End56 --> End57(( ))
        End57 --> End58(( ))
        End58 --> End59(( ))
        End59 --> End60(( ))
        End60 --> End61(( ))
        End61 --> End62(( ))
        End62 --> End63(( ))
        End63 --> End64(( ))
        End64 --> End65(( ))
        End65 --> End66(( ))
        End66 --> End67(( ))
        End67 --> End68(( ))
        End68 --> End69(( ))
        End69 --> End70(( ))
        End70 --> End71(( ))
        End71 --> End72(( ))
        End72 --> End73(( ))
        End73 --> End74(( ))
        End74 --> End75(( ))
        End75 --> End76(( ))
        End76 --> End77(( ))
        End77 --> End78(( ))
        End78 --> End79(( ))
        End79 --> End80(( ))
        End80 --> End81(( ))
        End81 --> End82(( ))
        End82 --> End83(( ))
        End83 --> End84(( ))
        End84 --> End85(( ))
        End85 --> End86(( ))
        End86 --> End87(( ))
        End87 --> End88(( ))
        End88 --> End89(( ))
        End89 --> End90(( ))
        End90 --> End91(( ))
        End91 --> End92(( ))
        End92 --> End93(( ))
        End93 --> End94(( ))
        End94 --> End95(( ))
        End95 --> End96(( ))
        End96 --> End97(( ))
        End97 --> End98(( ))
        End98 --> End99(( ))
        End99 --> End100(( ))
        End100 --> End101(( ))
        End101 --> End102(( ))
        End102 --> End103(( ))
        End103 --> End104(( ))
        End104 --> End105(( ))
        End105 --> End106(( ))
        End106 --> End107(( ))
        End107 --> End108(( ))
        End108 --> End109(( ))
        End109 --> End110(( ))
        End110 --> End111(( ))
        End111 --> End112(( ))
        End112 --> End113(( ))
        End113 --> End114(( ))
        End114 --> End115(( ))
        End115 --> End116(( ))
        End116 --> End117(( ))
        End117 --> End118(( ))
        End118 --> End119(( ))
        End119 --> End120(( ))
        End120 --> End121(( ))
        End121 --> End122(( ))
        End122 --> End123(( ))
        End123 --> End124(( ))
        End124 --> End125(( ))
        End125 --> End126(( ))
        End126 --> End127(( ))
        End127 --> End128(( ))
        End128 --> End129(( ))
        End129 --> End130(( ))
        End130 --> End131(( ))
        End131 --> End132(( ))
        End132 --> End133(( ))
        End133 --> End134(( ))
        End134 --> End135(( ))
        End135 --> End136(( ))
        End136 --> End137(( ))
        End137 --> End138(( ))
        End138 --> End139(( ))
        End139 --> End140(( ))
        End140 --> End141(( ))
        End141 --> End142(( ))
        End142 --> End143(( ))
        End143 --> End144(( ))
        End144 --> End145(( ))
        End145 --> End146(( ))
        End146 --> End147(( ))
        End147 --> End148(( ))
        End148 --> End149(( ))
        End149 --> End150(( ))
        End150 --> End151(( ))
        End151 --> End152(( ))
        End152 --> End153(( ))
        End153 --> End154(( ))
        End154 --> End155(( ))
        End155 --> End156(( ))
        End156 --> End157(( ))
        End157 --> End158(( ))
        End158 --> End159(( ))
        End159 --> End160(( ))
        End160 --> End161(( ))
        End161 --> End162(( ))
        End162 --> End163(( ))
        End163 --> End164(( ))
        End164 --> End165(( ))
        End165 --> End166(( ))
        End166 --> End167(( ))
        End167 --> End168(( ))
        End168 --> End169(( ))
        End169 --> End170(( ))
        End170 --> End171(( ))
        End171 --> End172(( ))
        End172 --> End173(( ))
        End173 --> End174(( ))
        End174 --> End175(( ))
        End175 --> End176(( ))
        End176 --> End177(( ))
        End177 --> End178(( ))
        End178 --> End179(( ))
        End179 --> End180(( ))
        End180 --> End181(( ))
        End181 --> End182(( ))
        End182 --> End183(( ))
        End183 --> End184(( ))
        End184 --> End185(( ))
        End185 --> End186(( ))
        End186 --> End187(( ))
        End187 --> End188(( ))
        End188 --> End189(( ))
        End189 --> End190(( ))
        End190 --> End191(( ))
        End191 --> End192(( ))
        End192 --> End193(( ))
        End193 --> End194(( ))
        End194 --> End195(( ))
        End195 --> End196(( ))
        End196 --> End197(( ))
        End197 --> End198(( ))
        End198 --> End199(( ))
        End199 --> End200(( ))
        End200 --> End201(( ))
        End201 --> End202(( ))
        End202 --> End203(( ))
        End203 --> End204(( ))
        End204 --> End205(( ))
        End205 --> End206(( ))
        End206 --> End207(( ))
        End207 --> End208(( ))
        End208 --> End209(( ))
        End209 --> End210(( ))
        End210 --> End211(( ))
        End211 --> End212(( ))
        End212 --> End213(( ))
        End213 --> End214(( ))
        End214 --> End215(( ))
        End215 --> End216(( ))
        End216 --> End217(( ))
        End217 --> End218(( ))
        End218 --> End219(( ))
        End219 --> End220(( ))
        End220 --> End221(( ))
        End221 --> End222(( ))
        End222 --> End223(( ))
        End223 --> End224(( ))
        End224 --> End225(( ))
        End225 --> End226(( ))
        End226 --> End227(( ))
        End227 --> End228(( ))
        End228 --> End229(( ))
        End229 --> End230(( ))
        End230 --> End231(( ))
        End231 --> End232(( ))
        End232 --> End233(( ))
        End233 --> End234(( ))
        End234 --> End235(( ))
        End235 --> End236(( ))
        End236 --> End237(( ))
        End237 --> End238(( ))
        End238 --> End239(( ))
        End239 --> End240(( ))
        End240 --> End241(( ))
        End241 --> End242(( ))
        End242 --> End243(( ))
        End243 --> End244(( ))
        End244 --> End245(( ))
        End245 --> End246(( ))
        End246 --> End247(( ))
        End247 --> End248(( ))
        End2
```

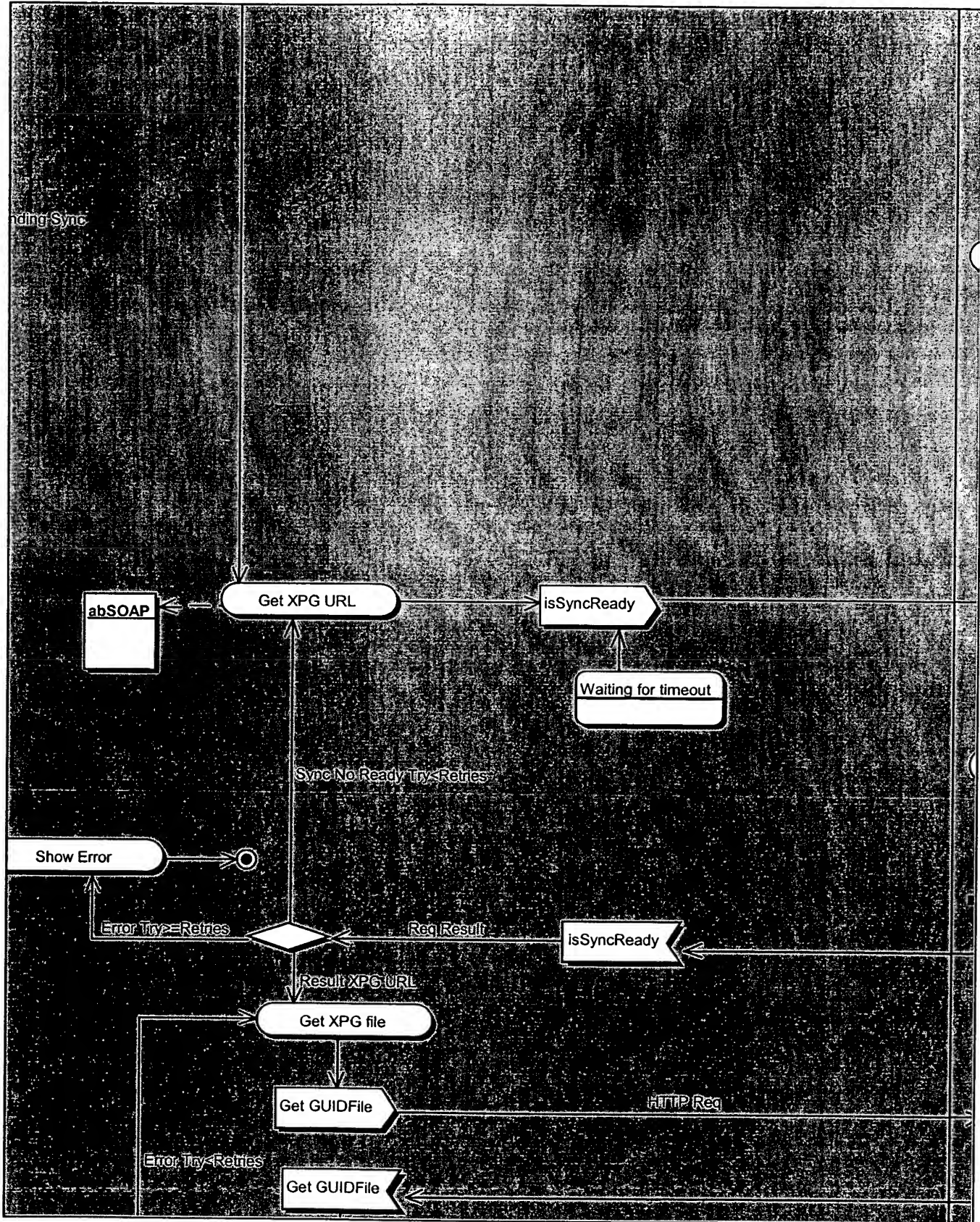
3.3, (2)txSync

4.1, (2)txSync



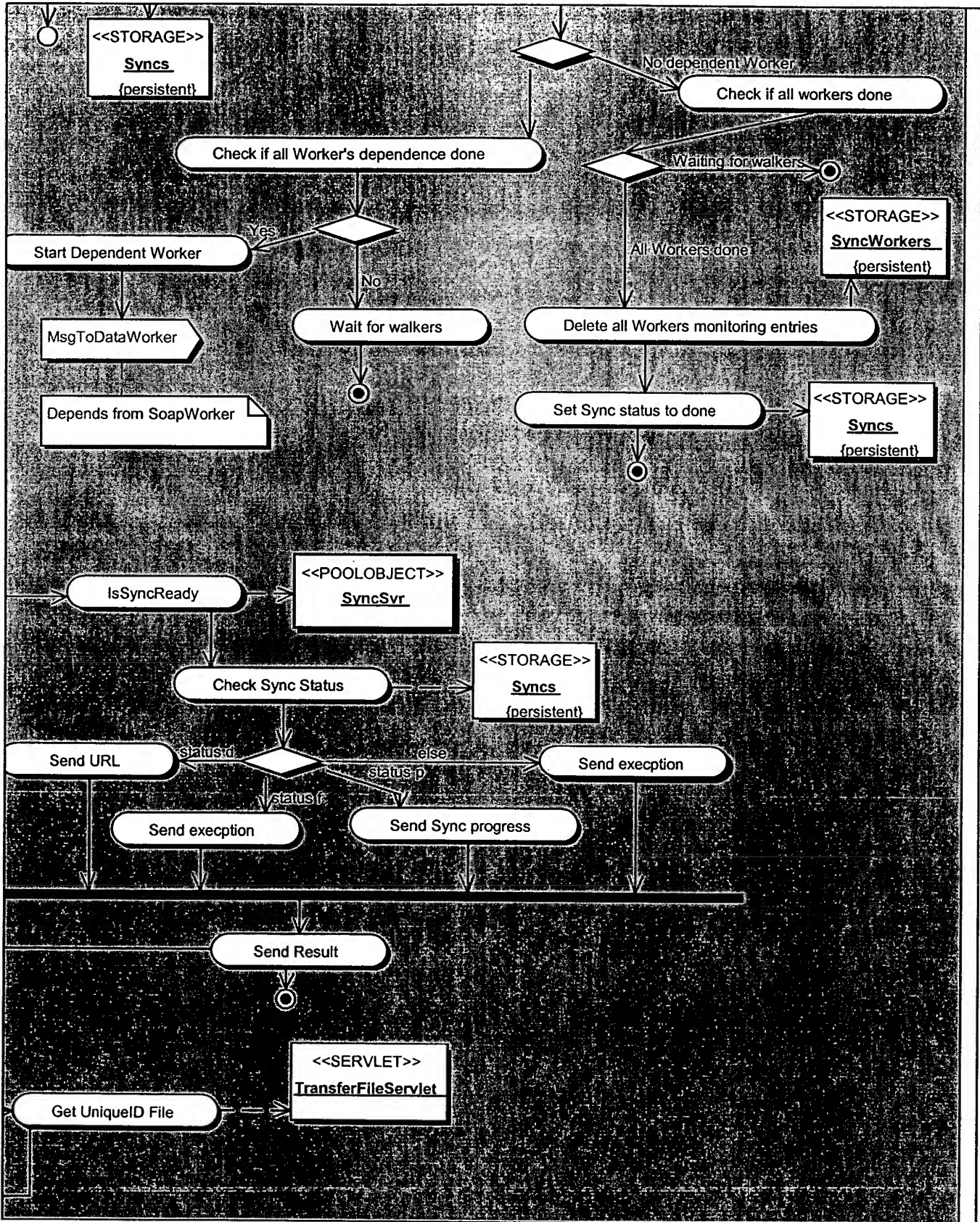
4.1, (2)txSync

4.2, (2)txSync



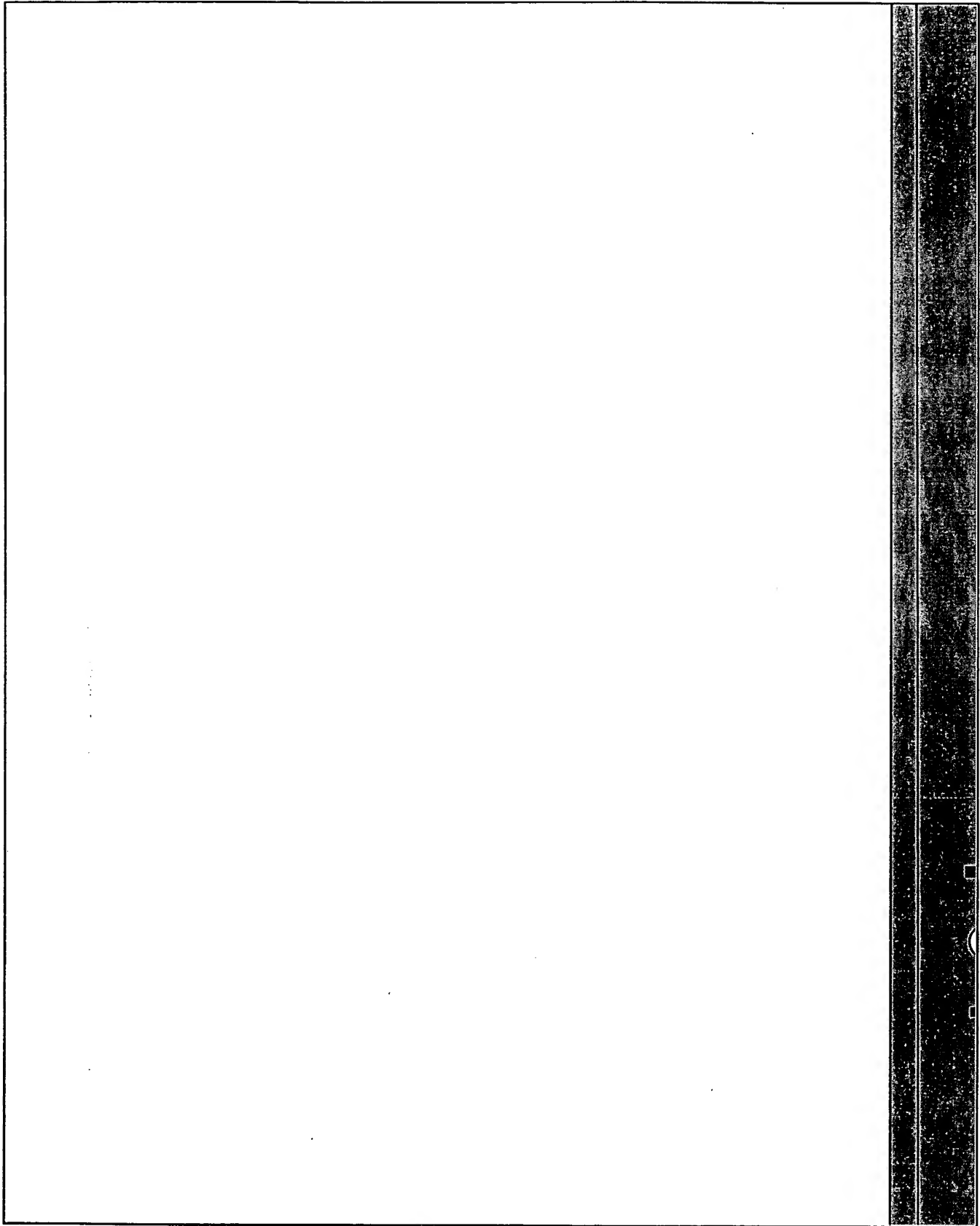
4.2, (2)txSync

4.3, (2)txSync



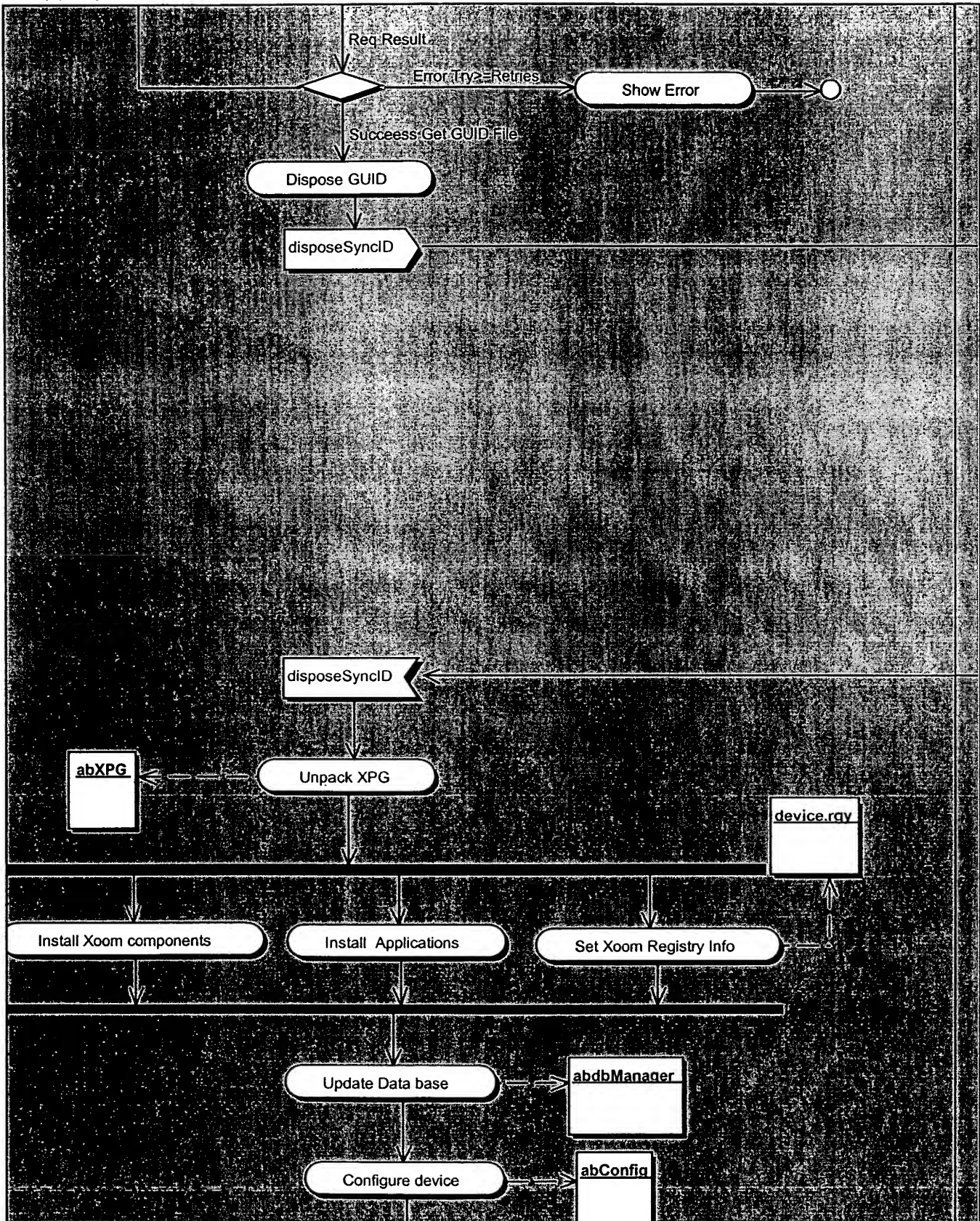
4.3, (2)txSync

5.1, (2)txSync



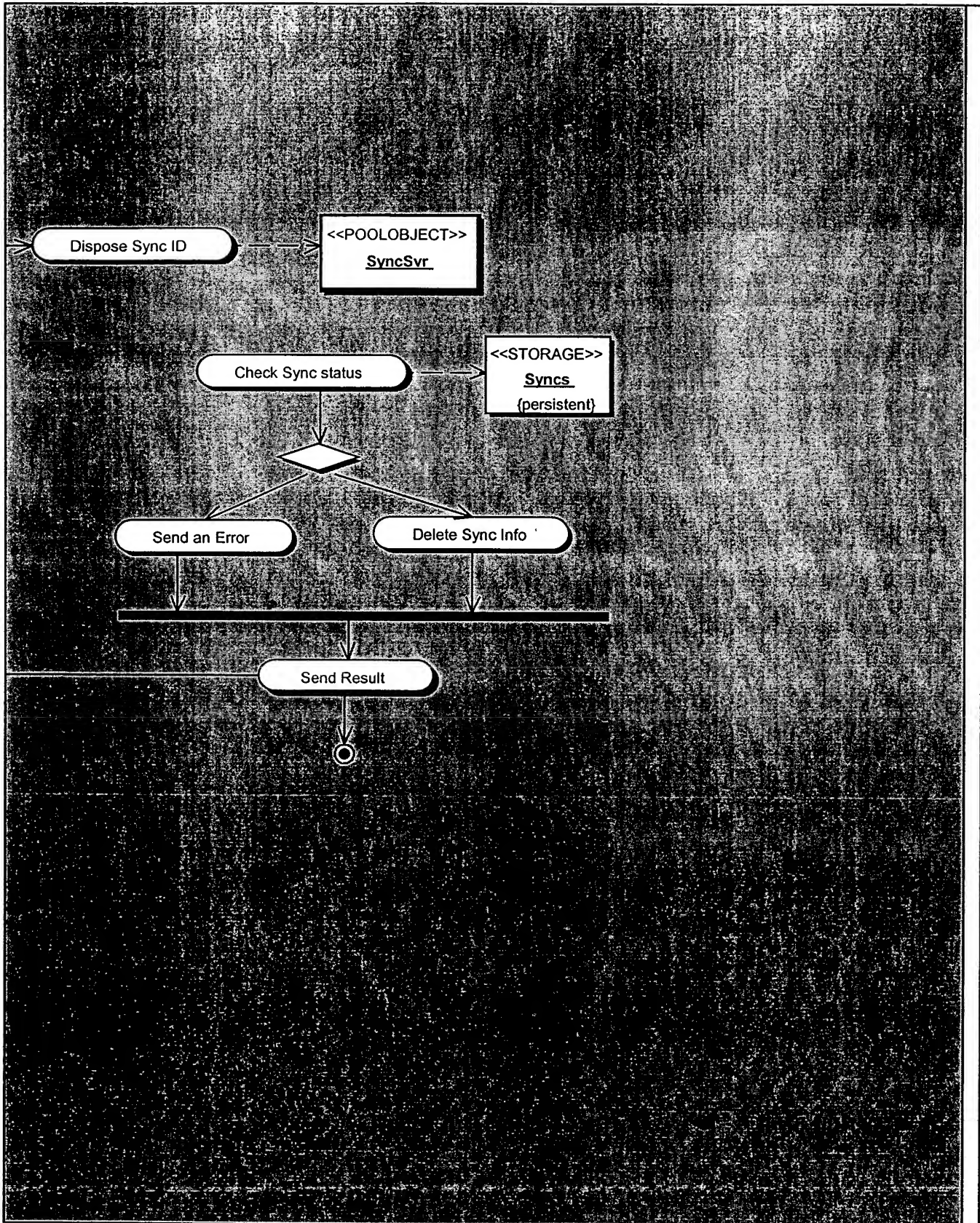
5.1, (2)txSync

5.2, (2)txSync



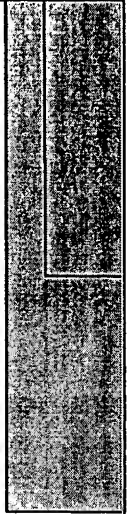
5.2, (2)txSync

5.3, (2)txSync



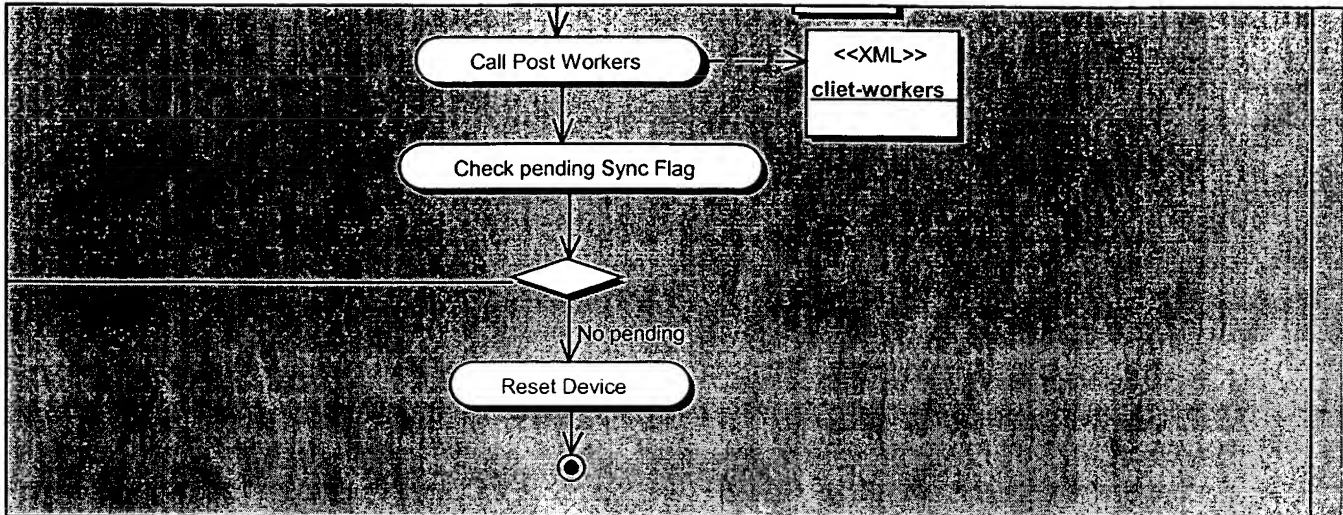
5.3, (2)txSync

6.1, (2)txSync



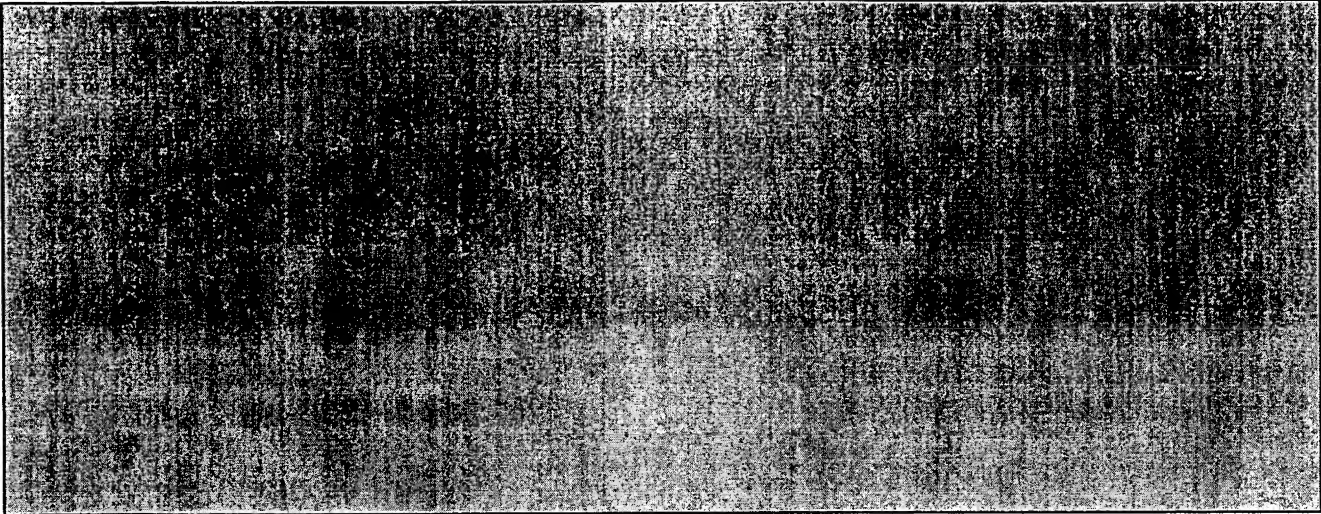
6.1, (2)txSync

6.2, (2)txSync



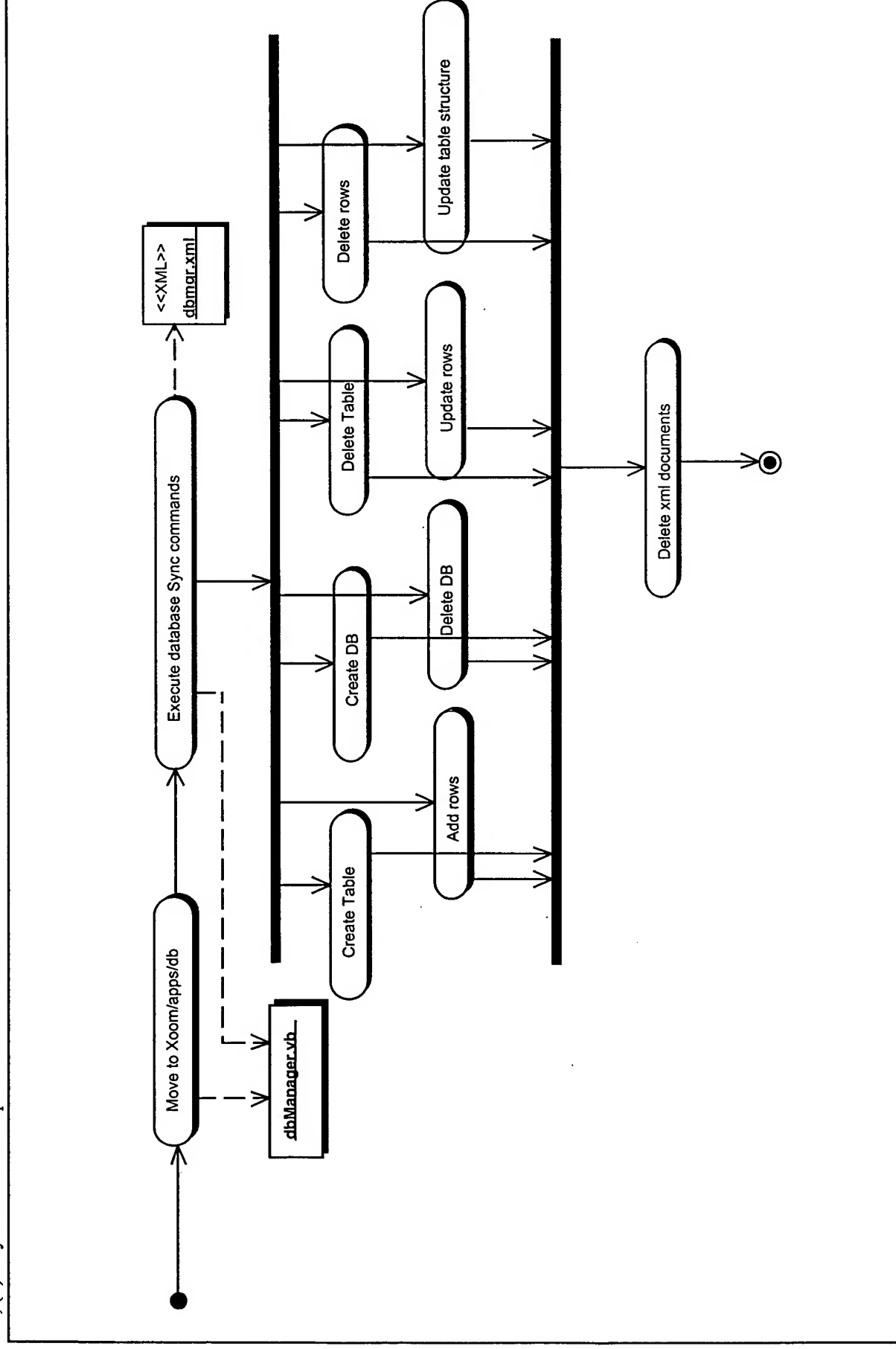
6.2, (2)txSync

6.3, (2)txSync



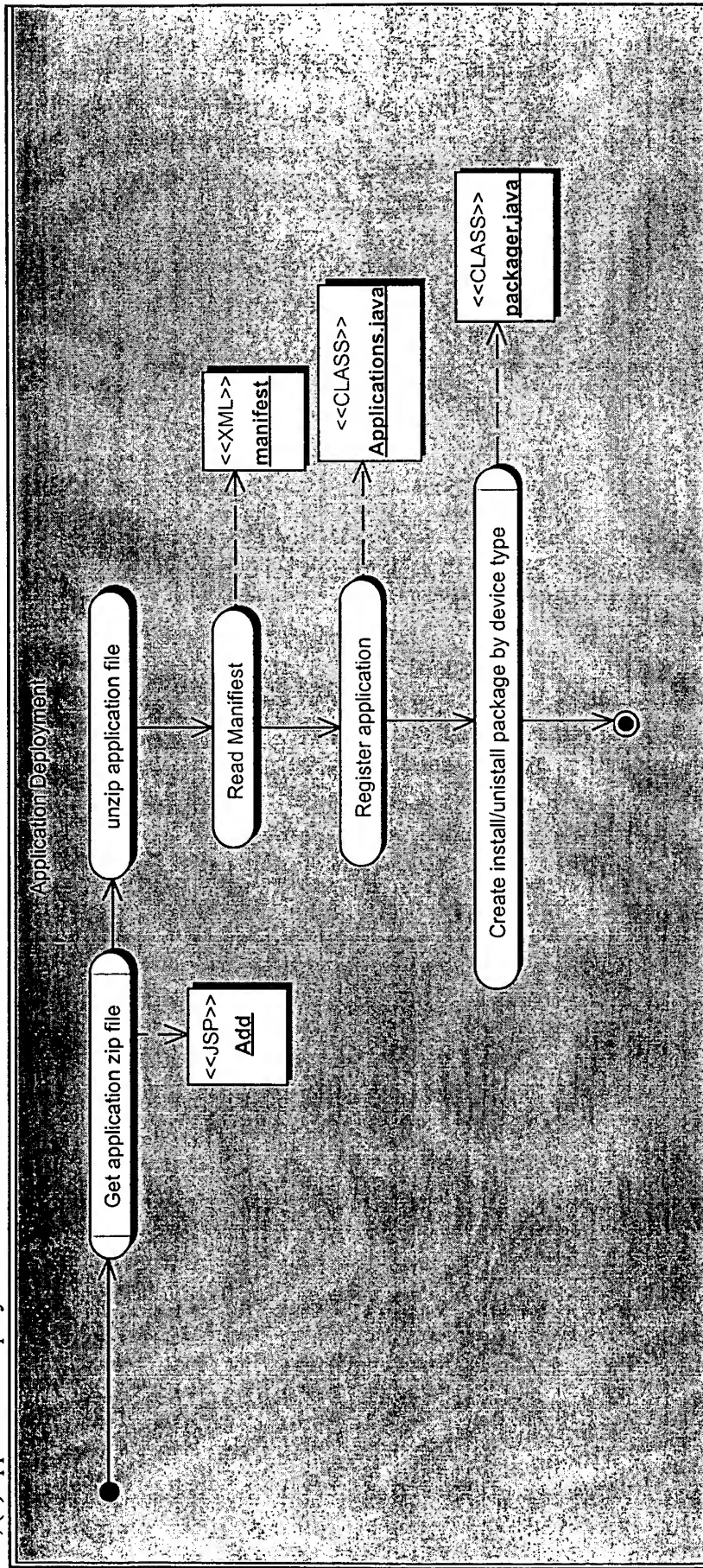
6.3, (2)txSync

1.1, (2)txSync Database Update



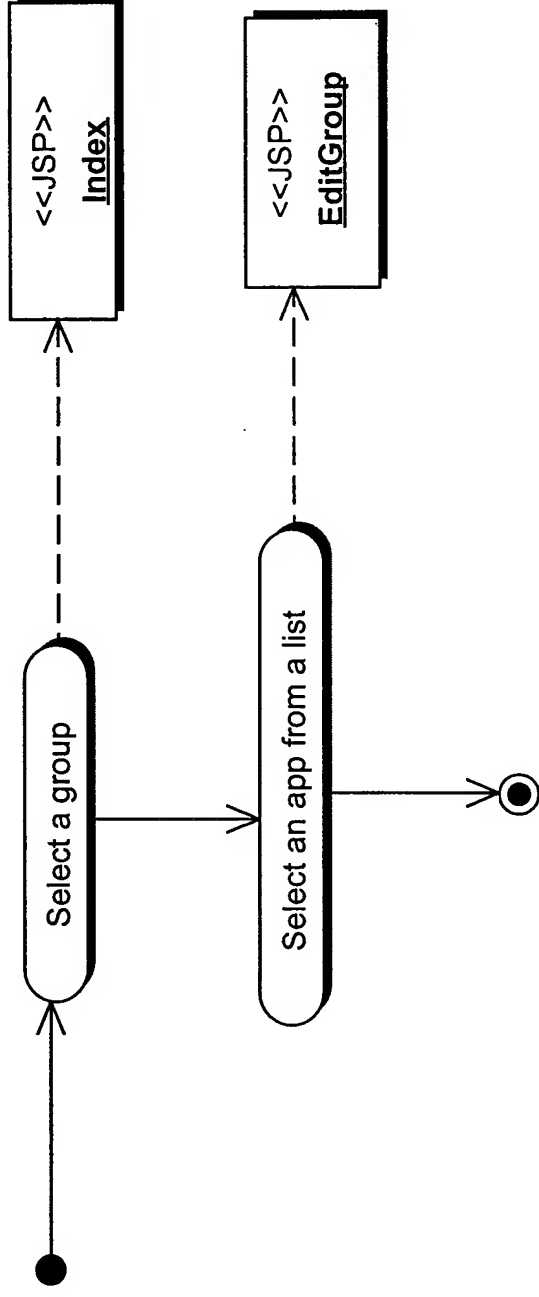
1.1, (2)txSync Database Update

1.1, (3)Application deployment



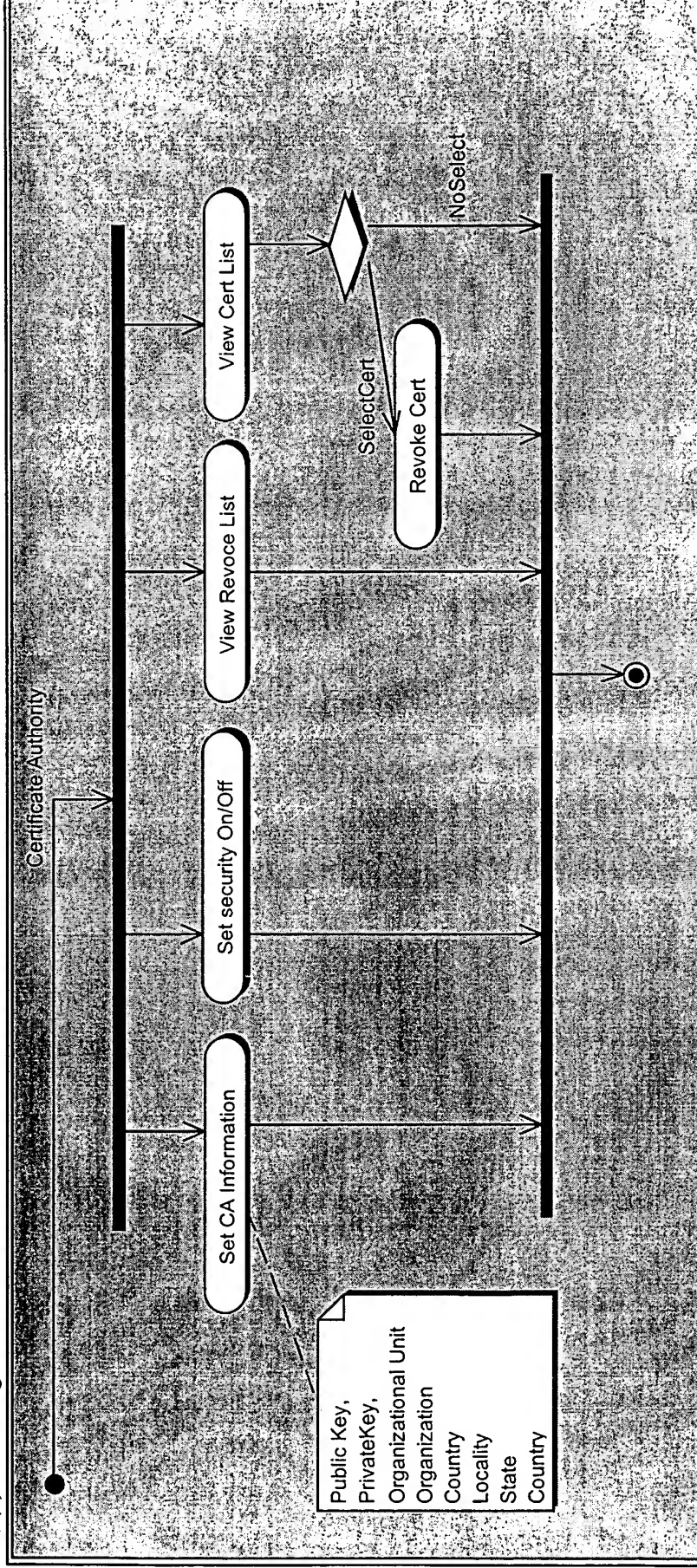
1.1, (3)Application deployment

1.1, (3)Assign application to group



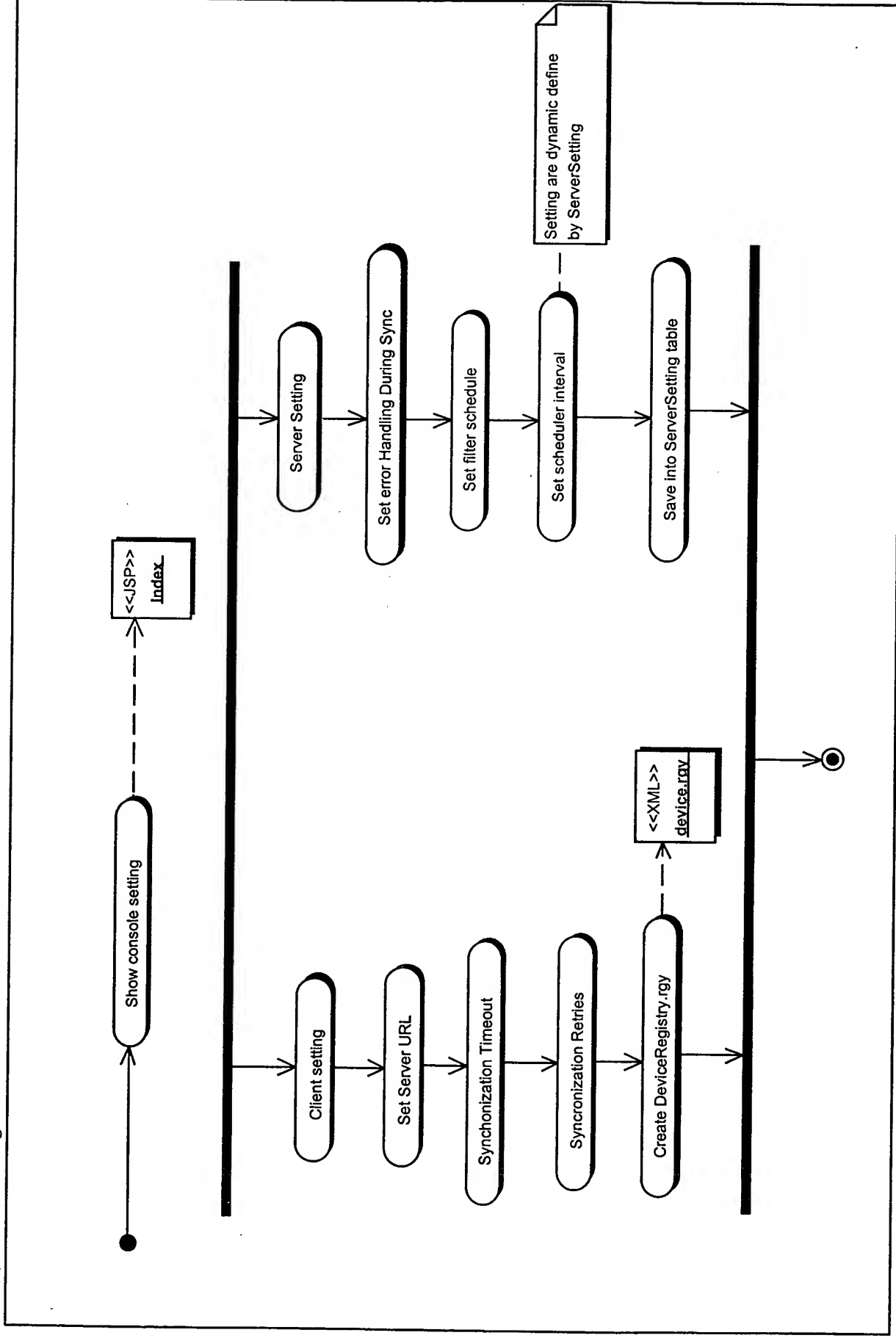
1.1, (3)Assign application to group

1.1, (3)CA Setting



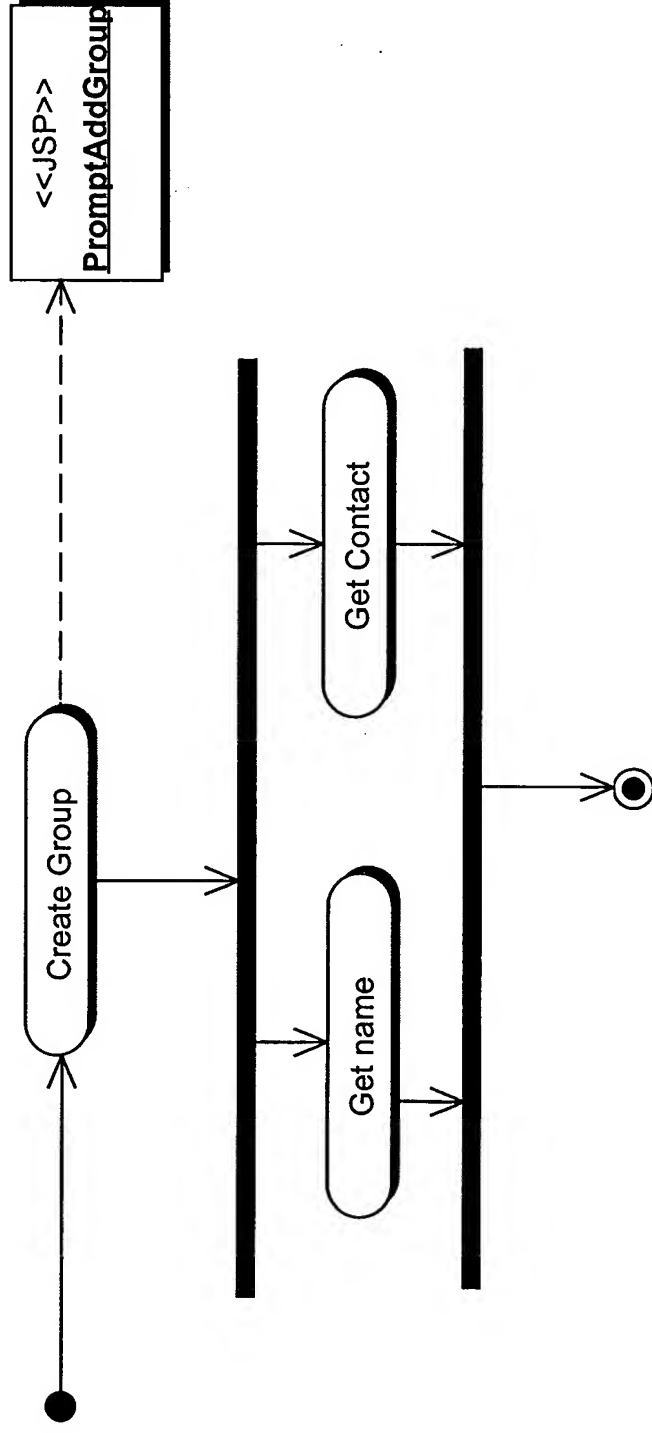
1.1, (3)CA Setting

1.1, (3) Console setting



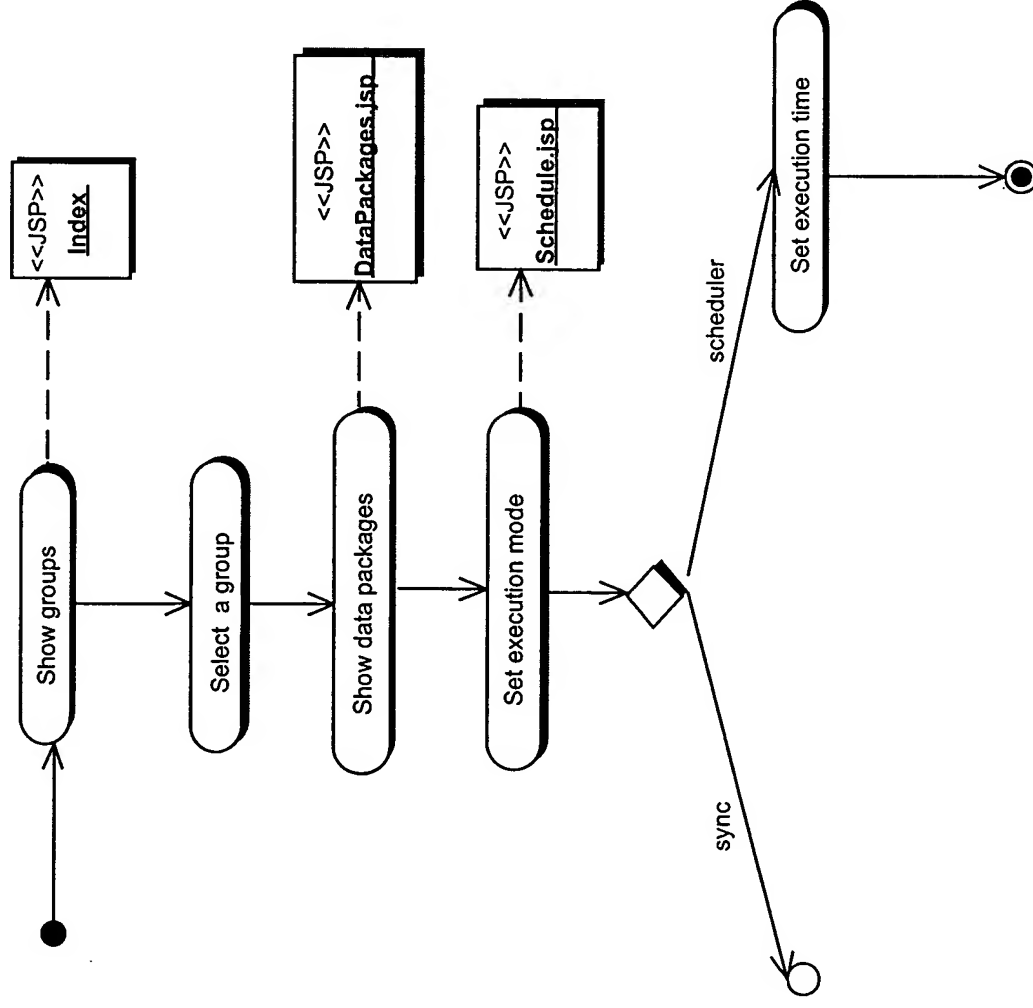
1.1, (3) Console setting

1.1, (3)Create group



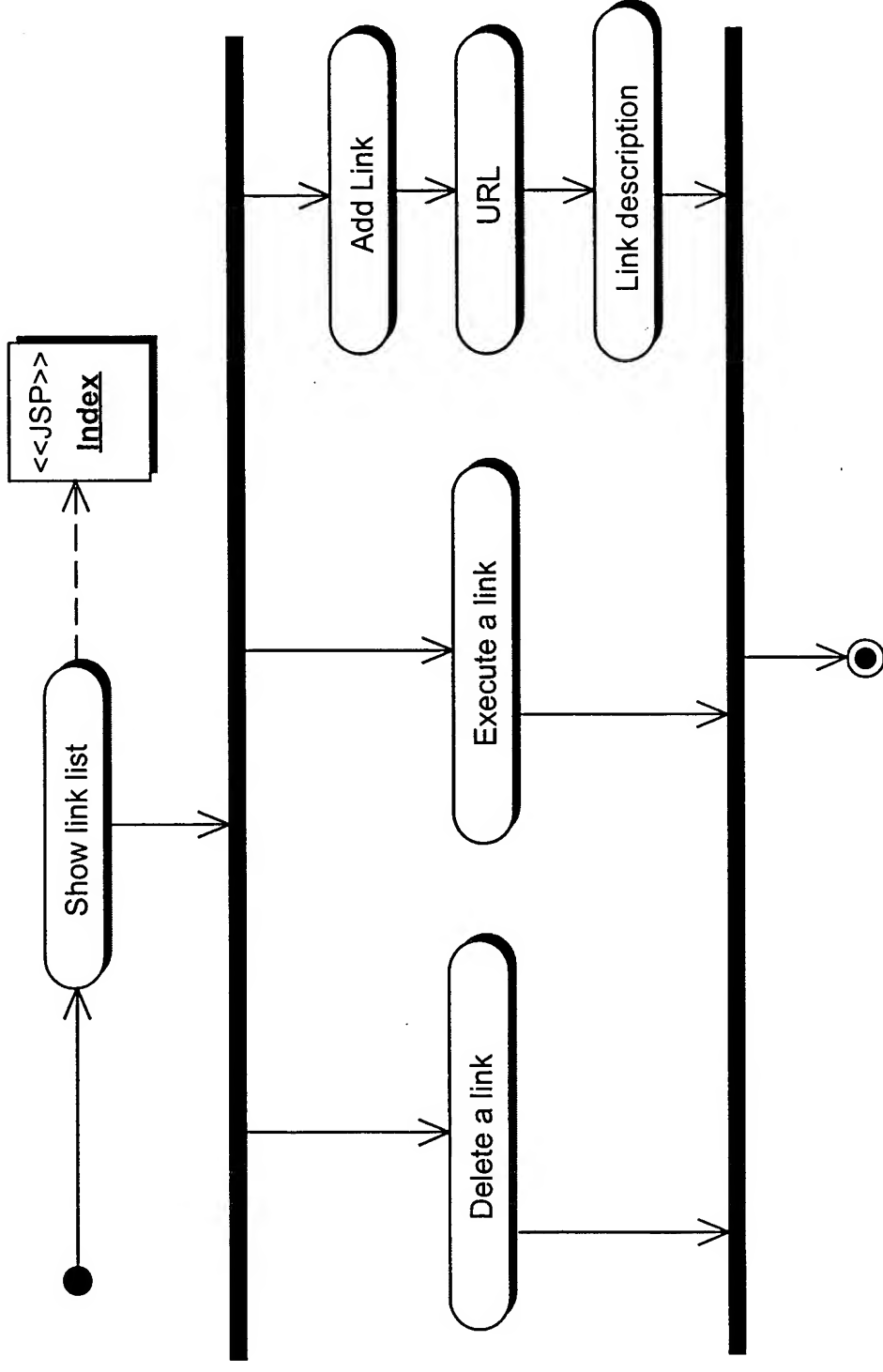
1.1, (3)Create group

1.1, (3)Data preparation



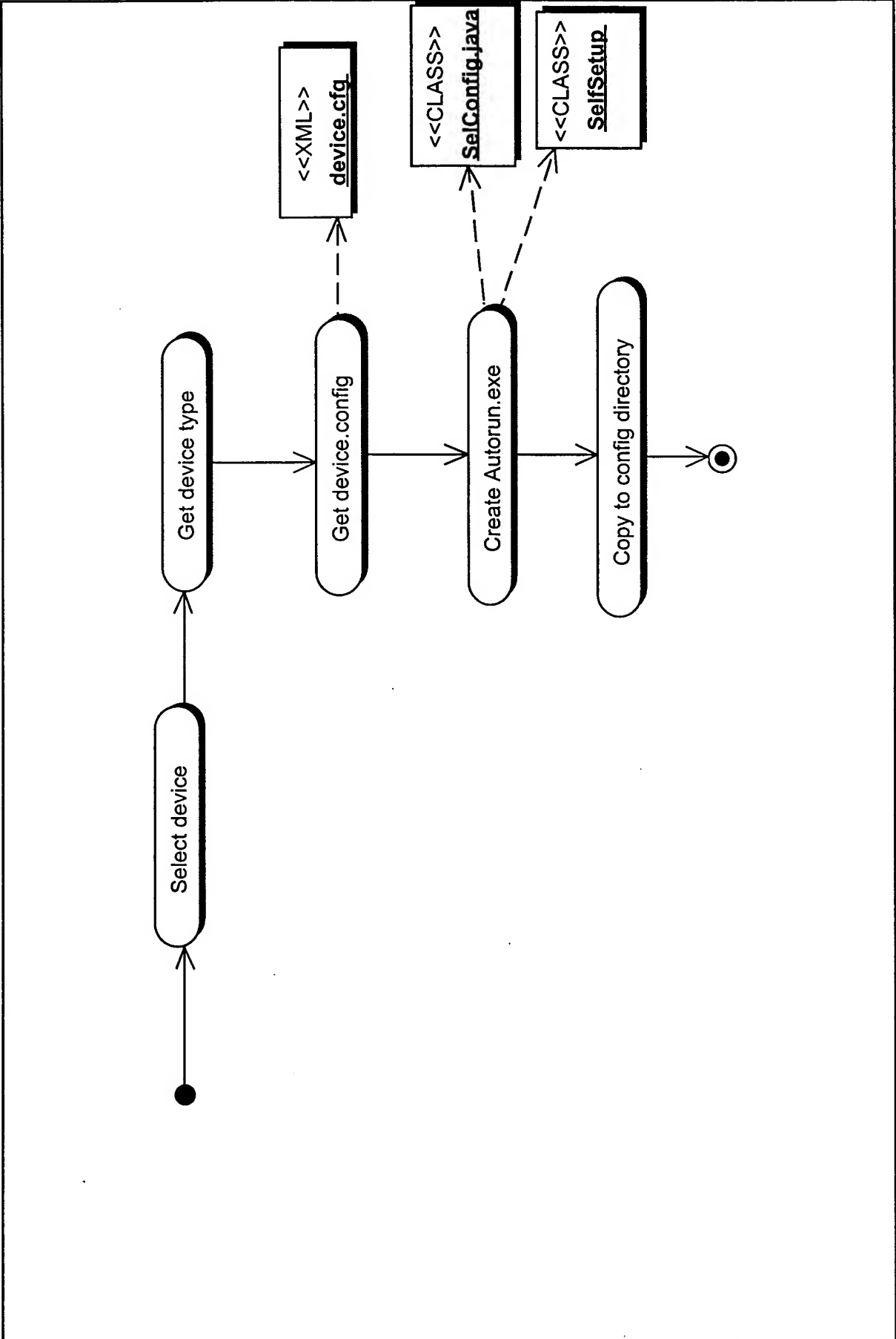
1.1, (3)Data preparation

1.1, (3) Define applications links



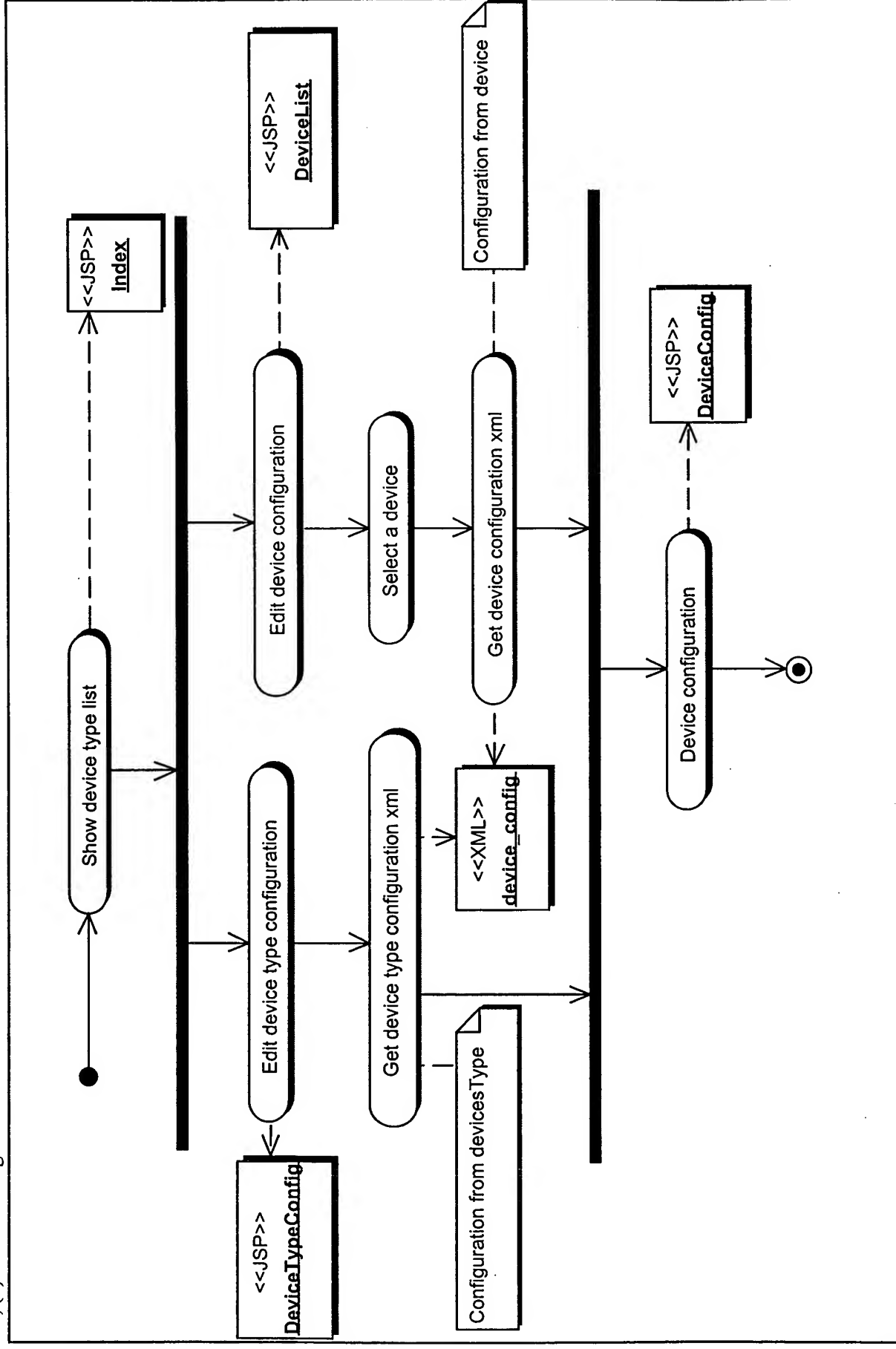
1.1, (3) Define applications links

1.1, (3)Device auto connection configuration



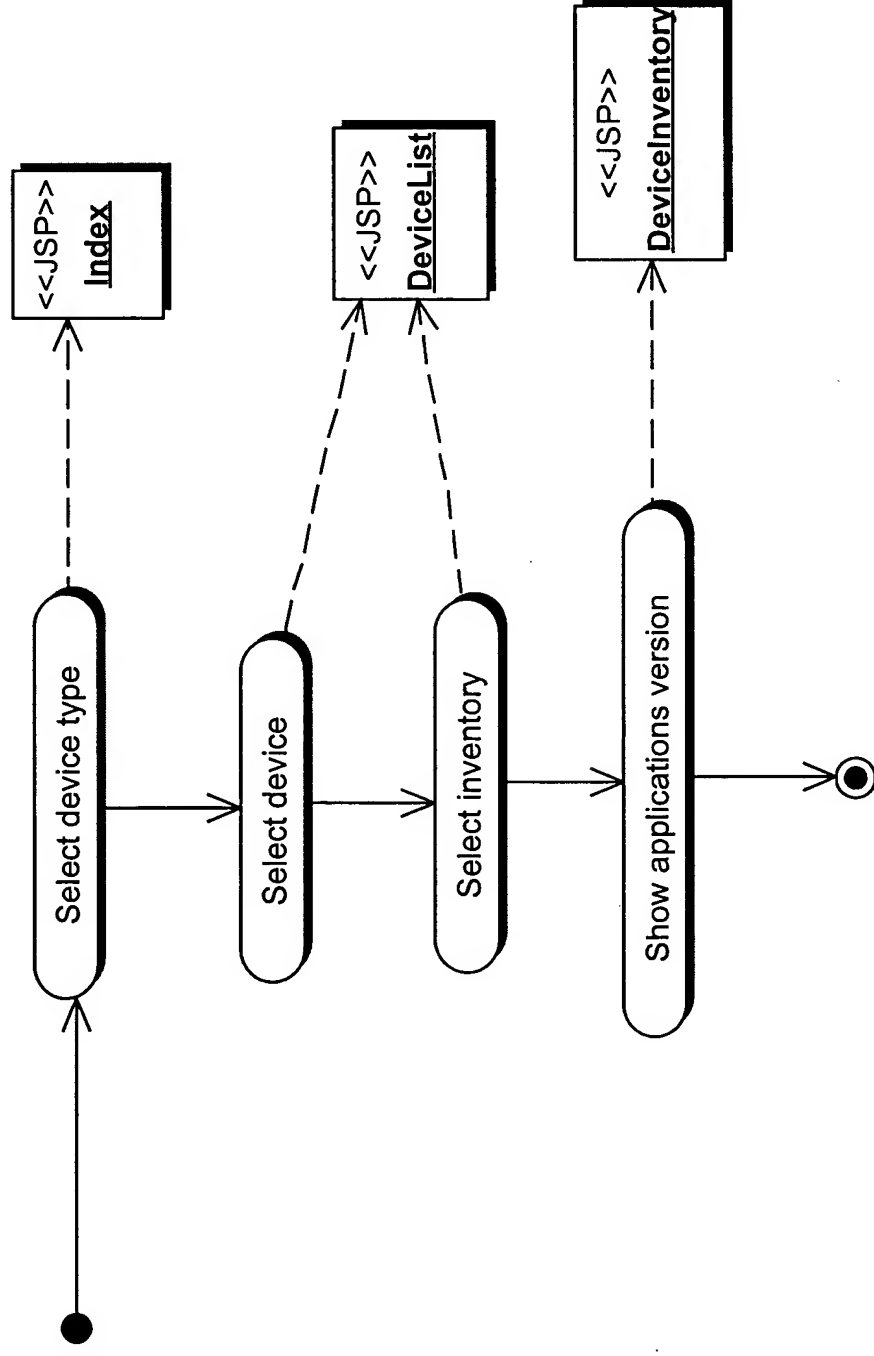
1.1, (3)Device auto connection configuration

1.1, (3)Device configuration



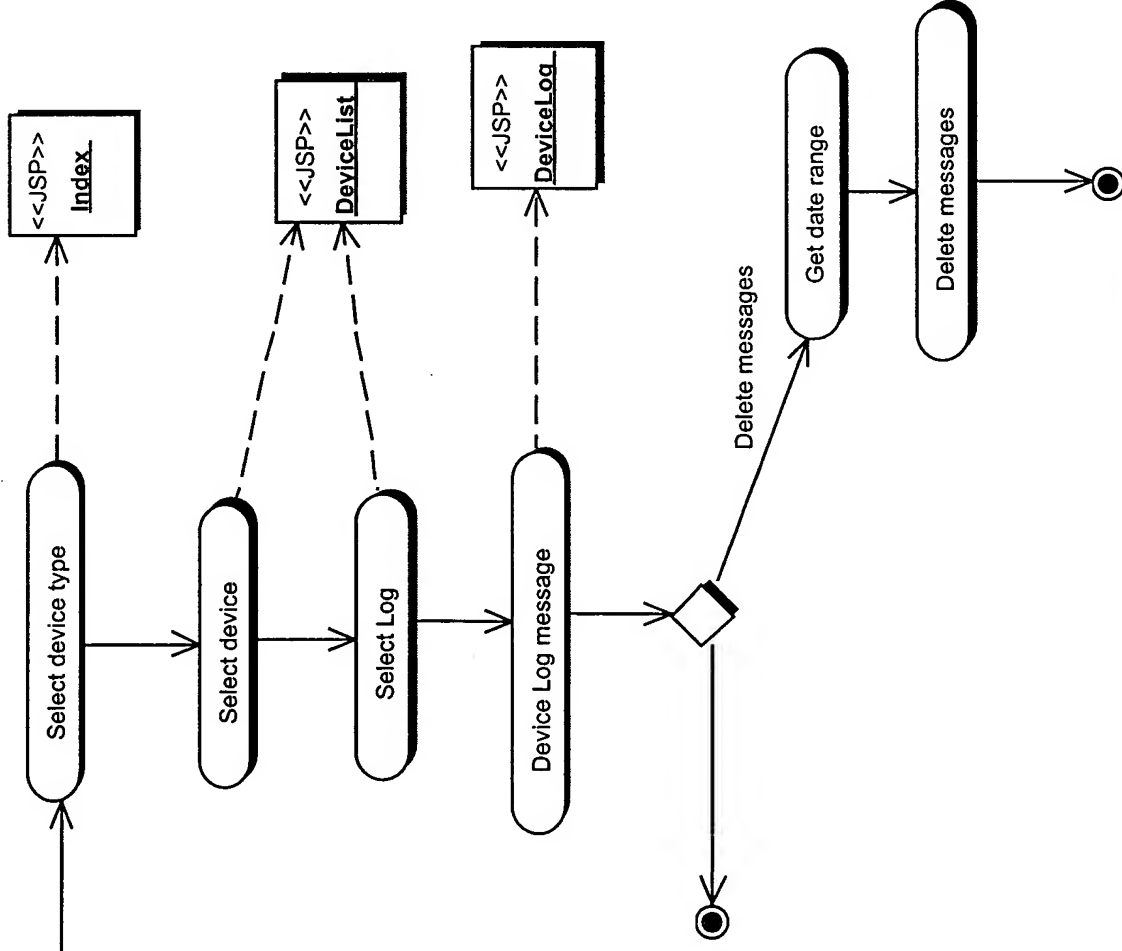
1.1, (3)Device configuration

1.1, (3)Device inventory



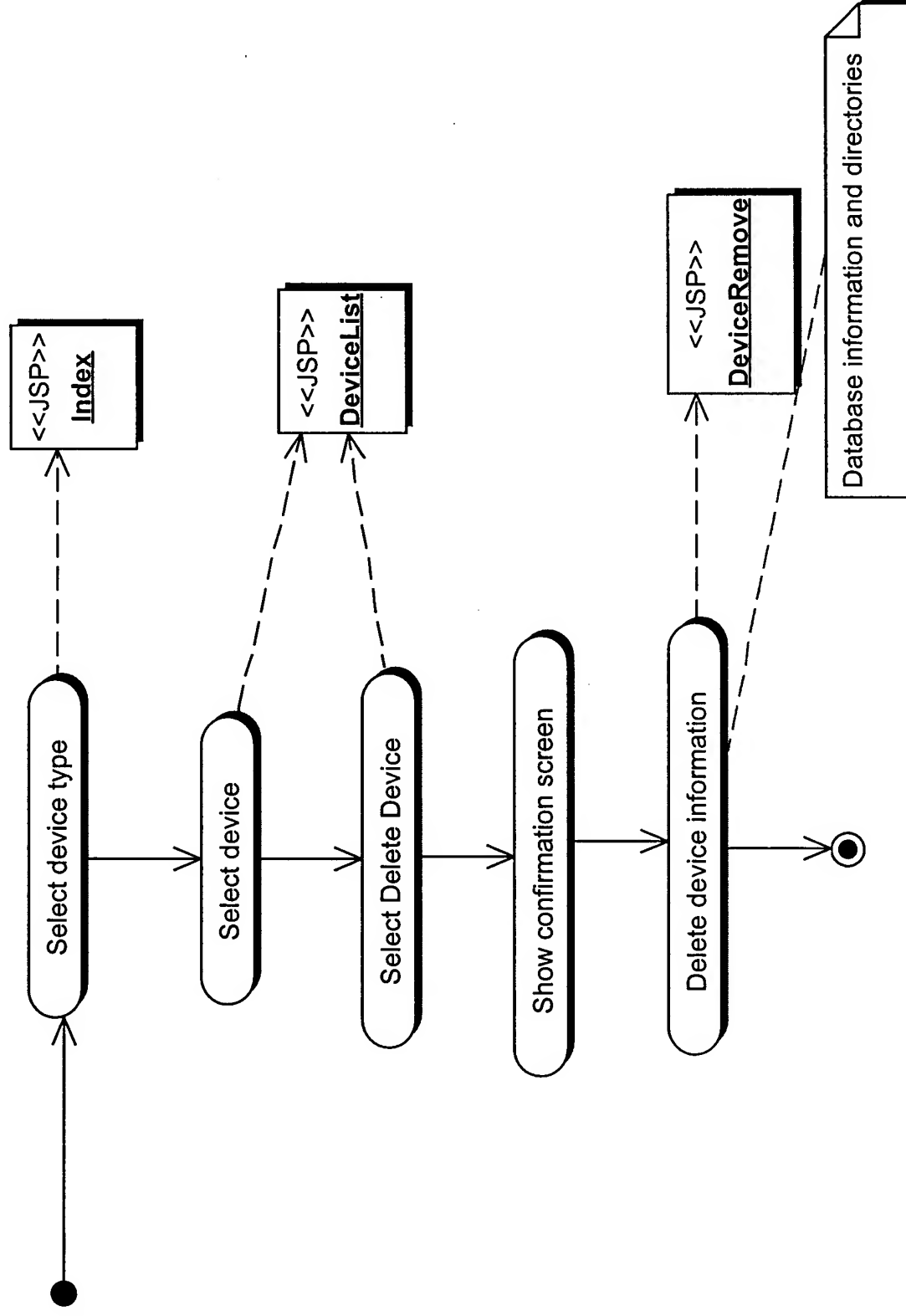
1.1, (3)Device inventory

1.1, (3)Device Log



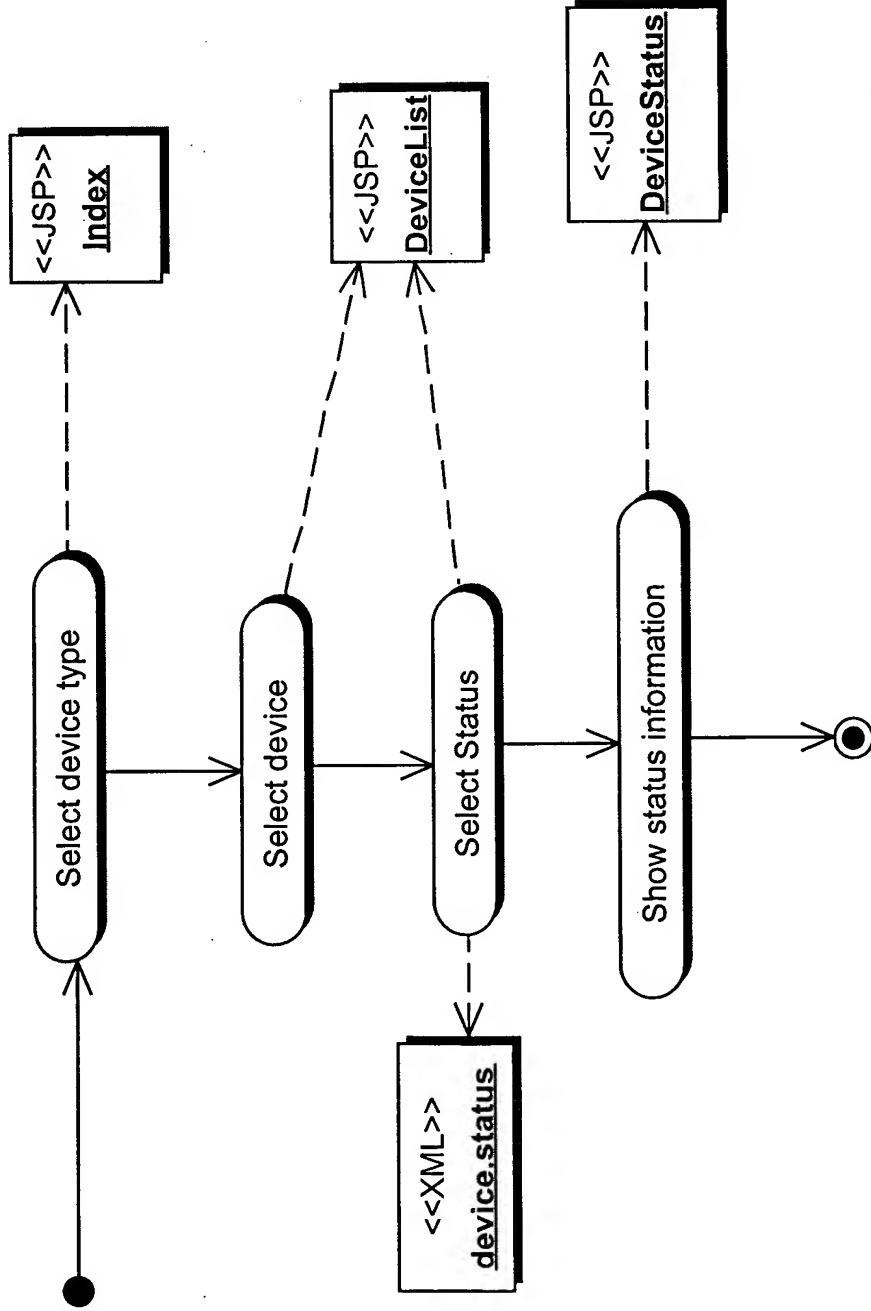
1.1, (3)Device Log

1.1, (3)Device removal



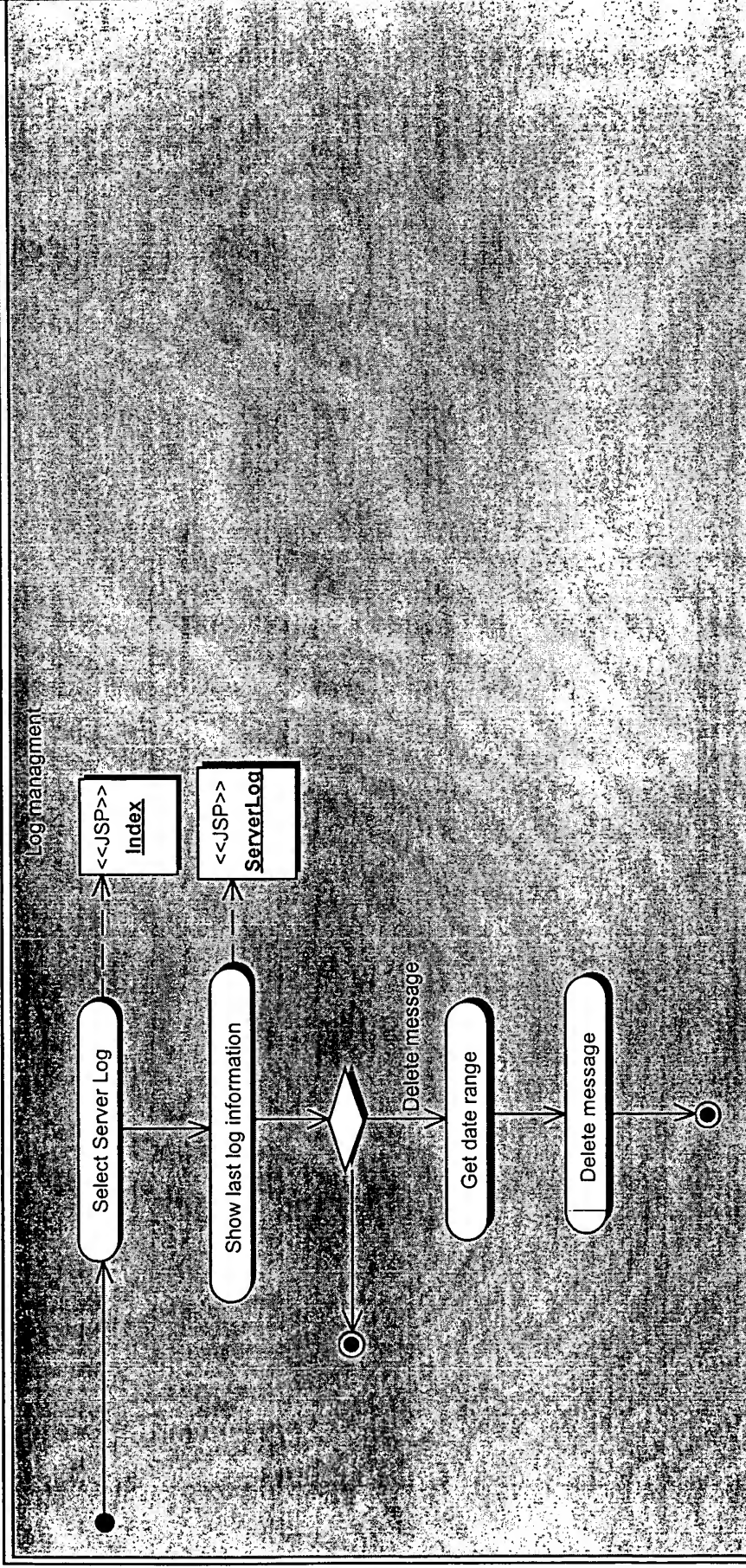
1.1, (3)Device removal

1.1, (3)Device status



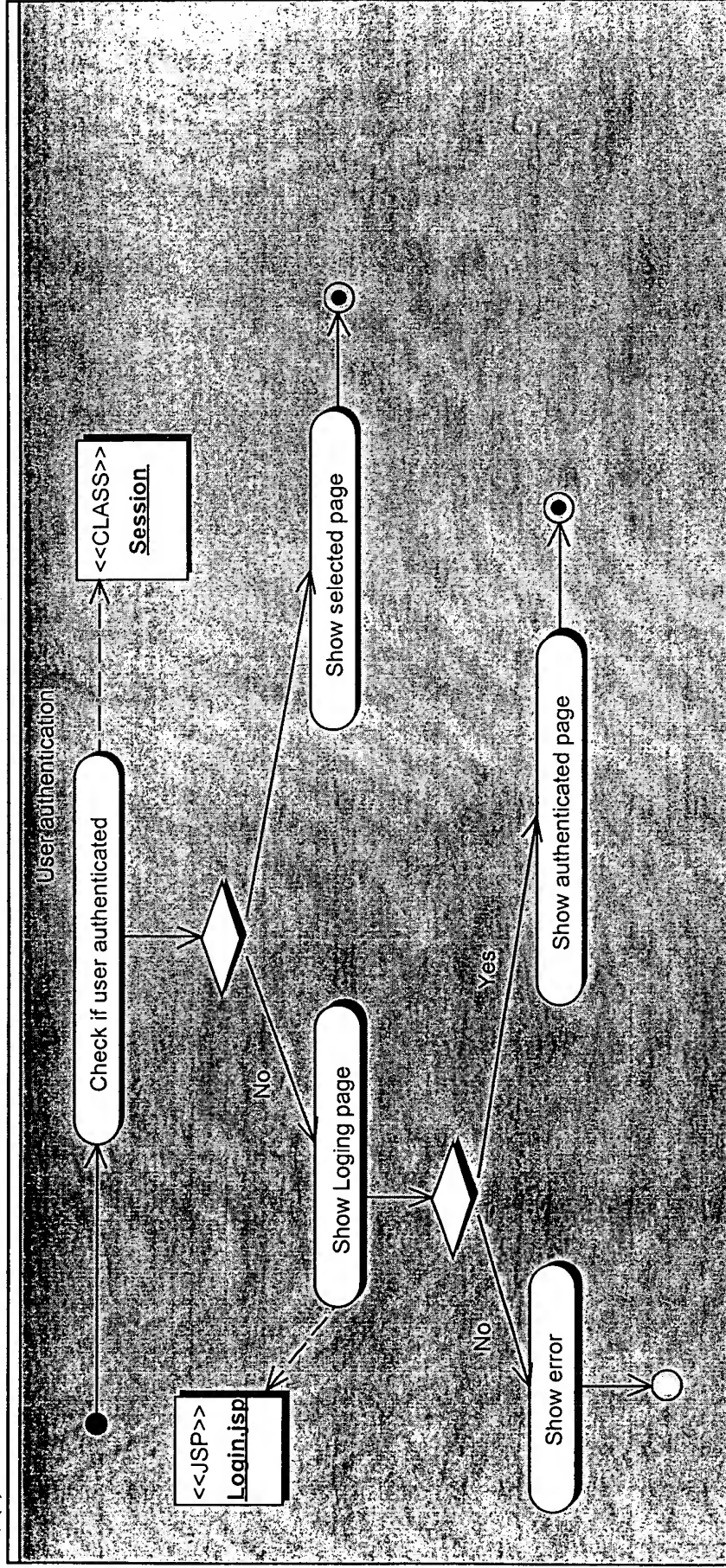
1.1, (3)Device status

1.1, (3)Log Managment



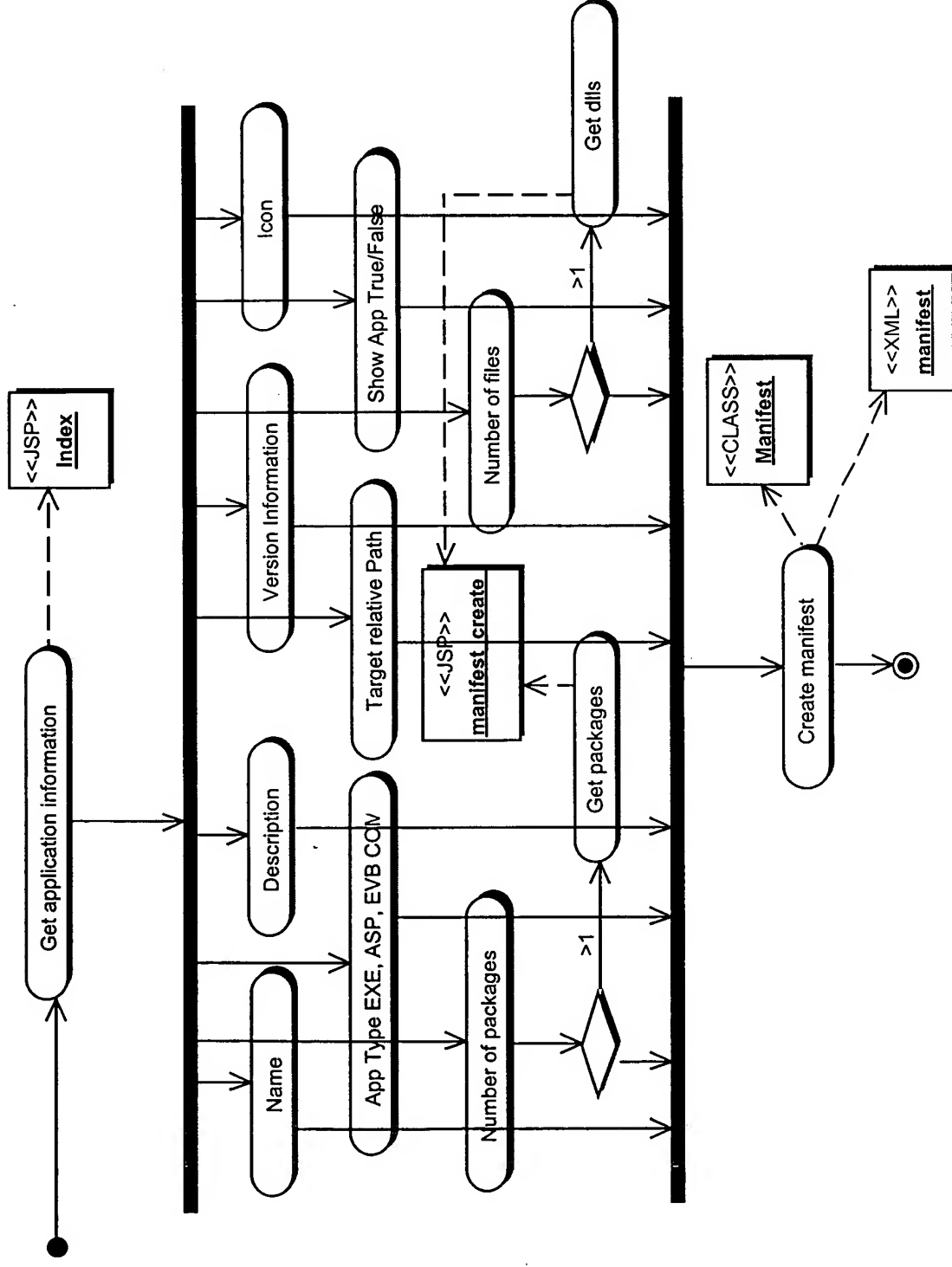
1.1, (3)Log Managment

1.1, (3)User Authentication



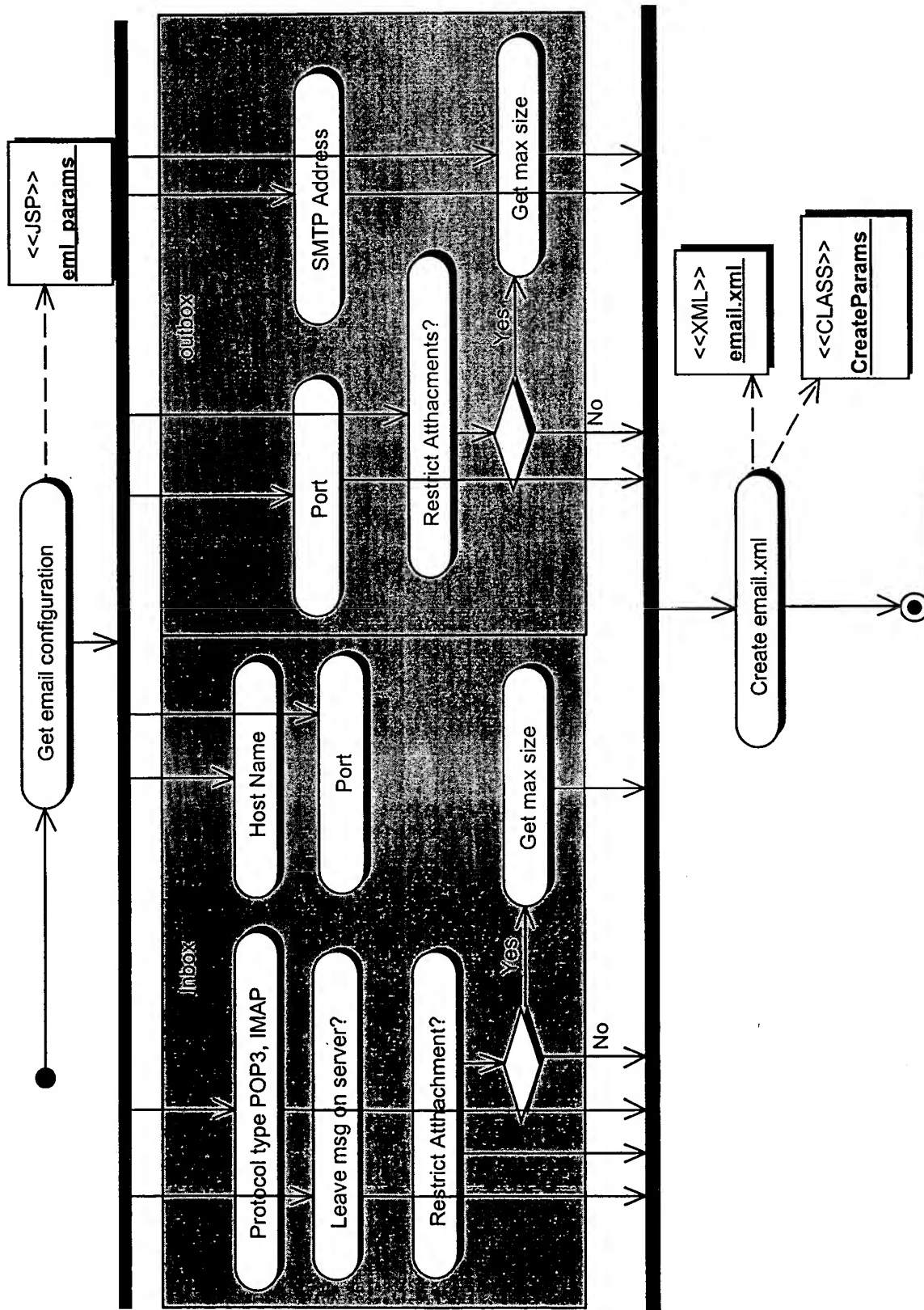
1.1, (3)User Authentication

1.1, (4)Create application manifest



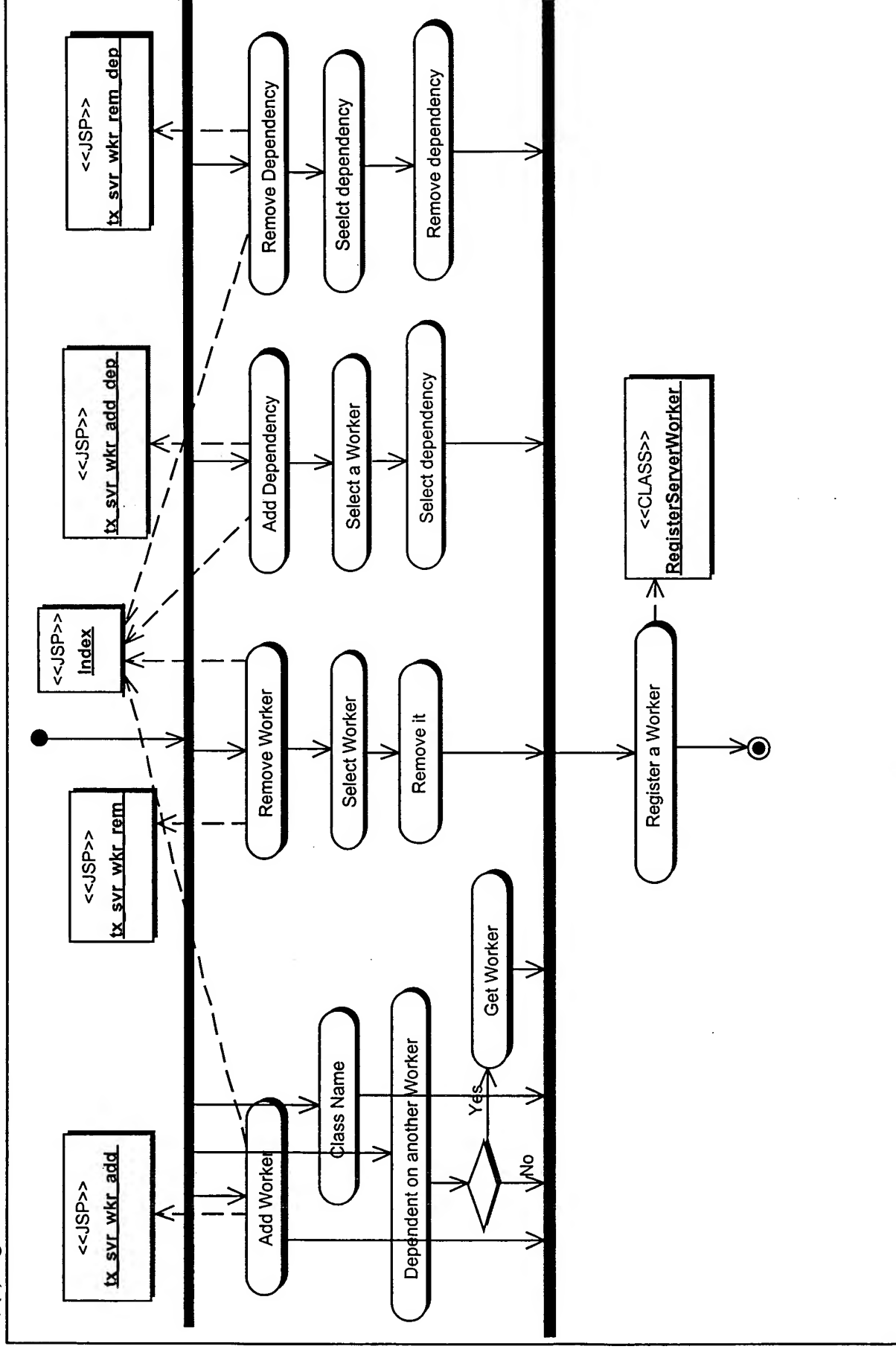
1.1, (4)Create application manifest

1.1, (4)Email setting



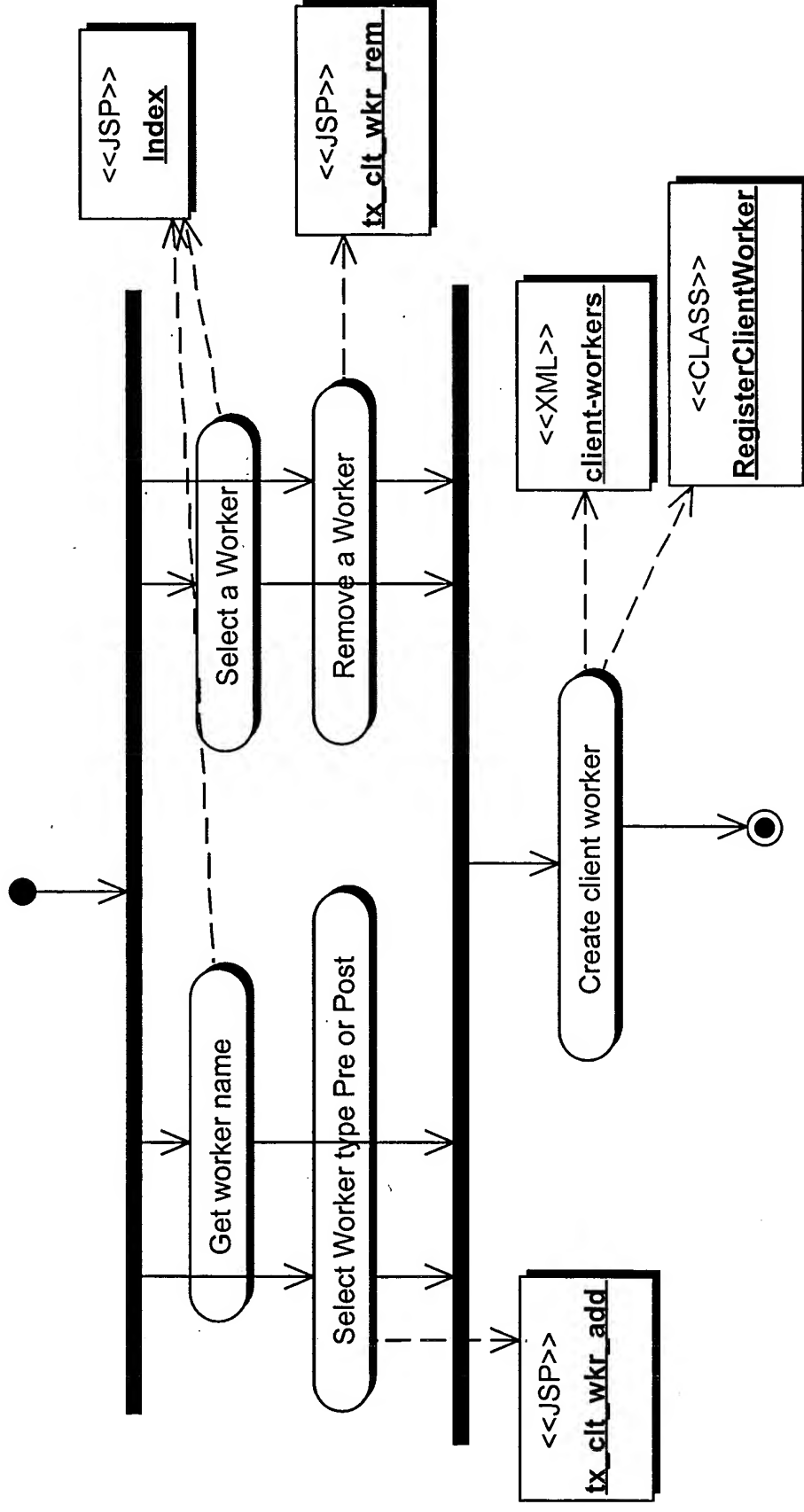
1.1, (4)Email setting

1.1, (4)Register Server Workers



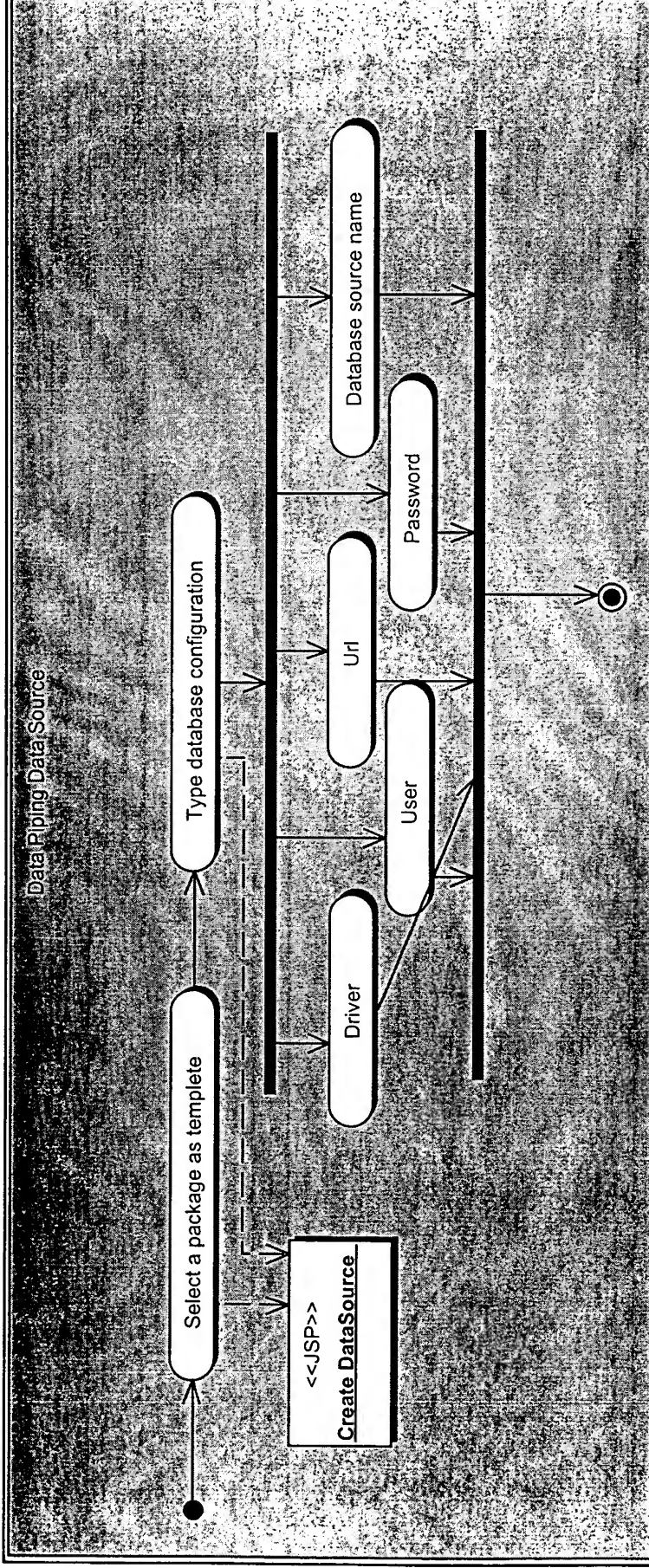
1.1, (4)Register Server Workers

1.1, (4)Regsiter Client Workers



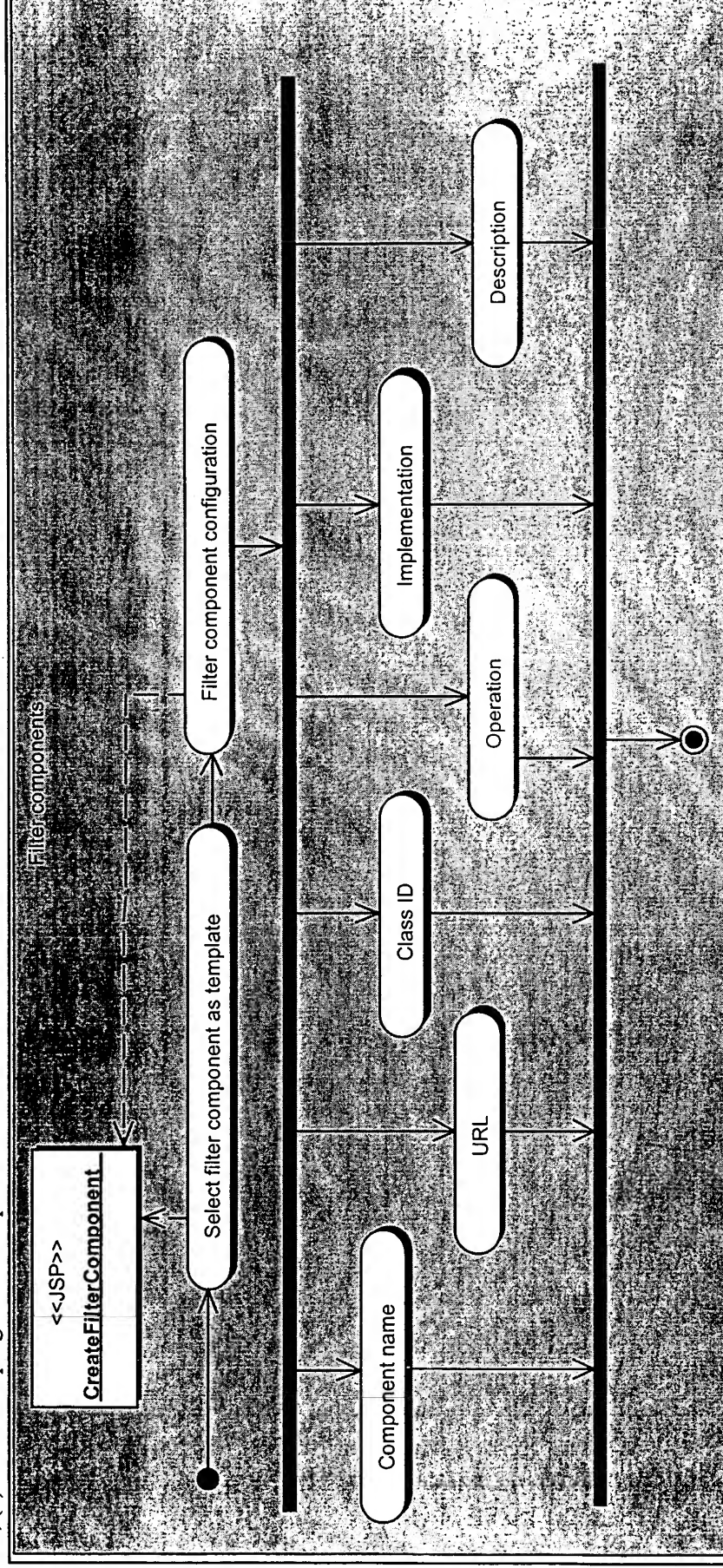
1.1, (4)Regsiter Client Workers

1.1.1, (4)Set Data Piping Data Source



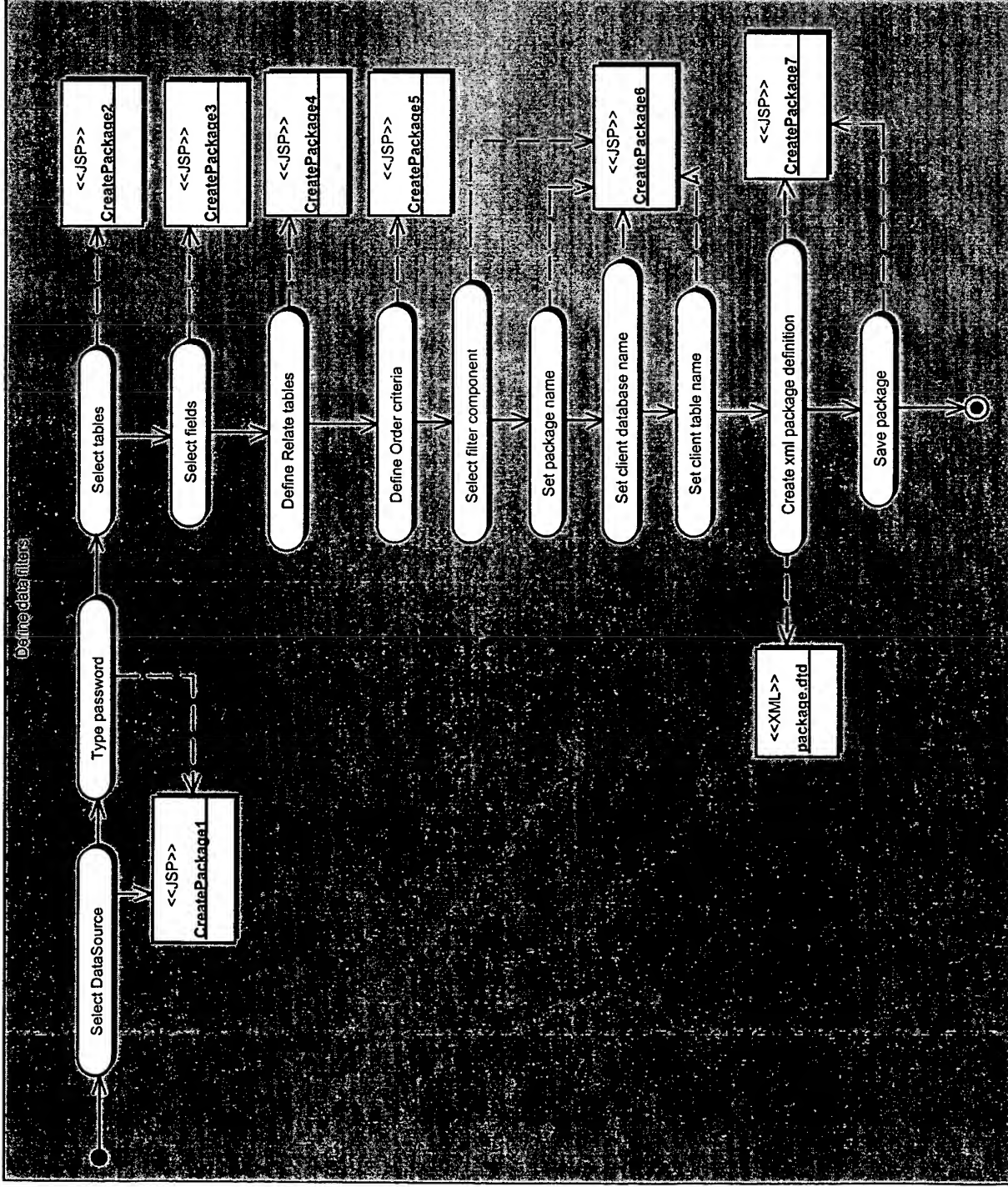
1.1.1, (4)Set Data Piping Data Source

1.1, (4)Set Data Piping Filter components

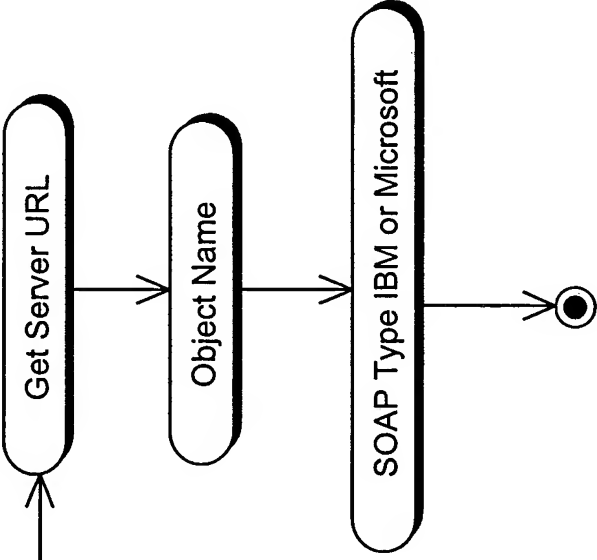


1.1, (4)Set Data Piping Filter components

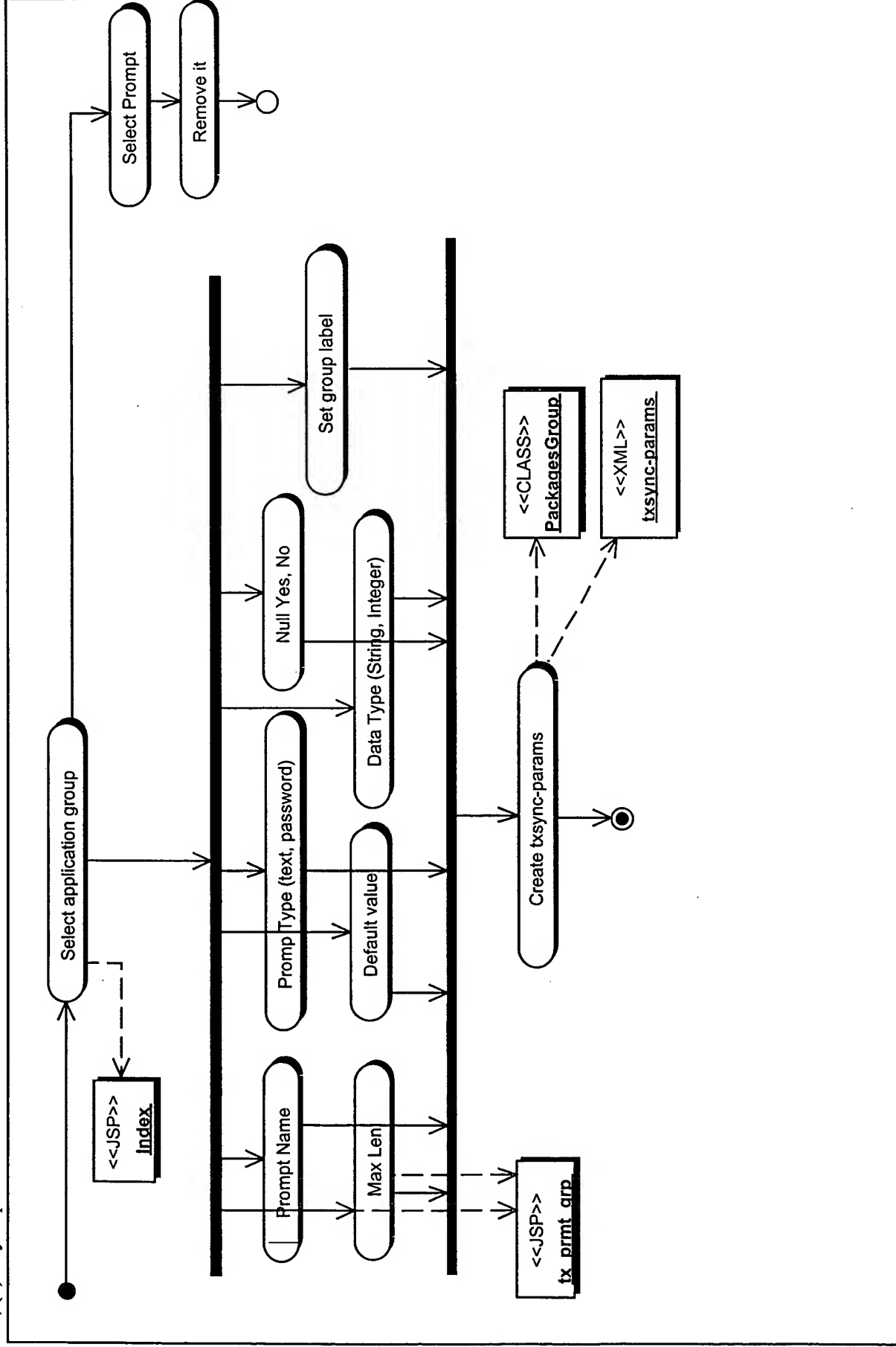
1.1, (4)Set Data Piping filters



1.1, (4)Set Data Piping filters

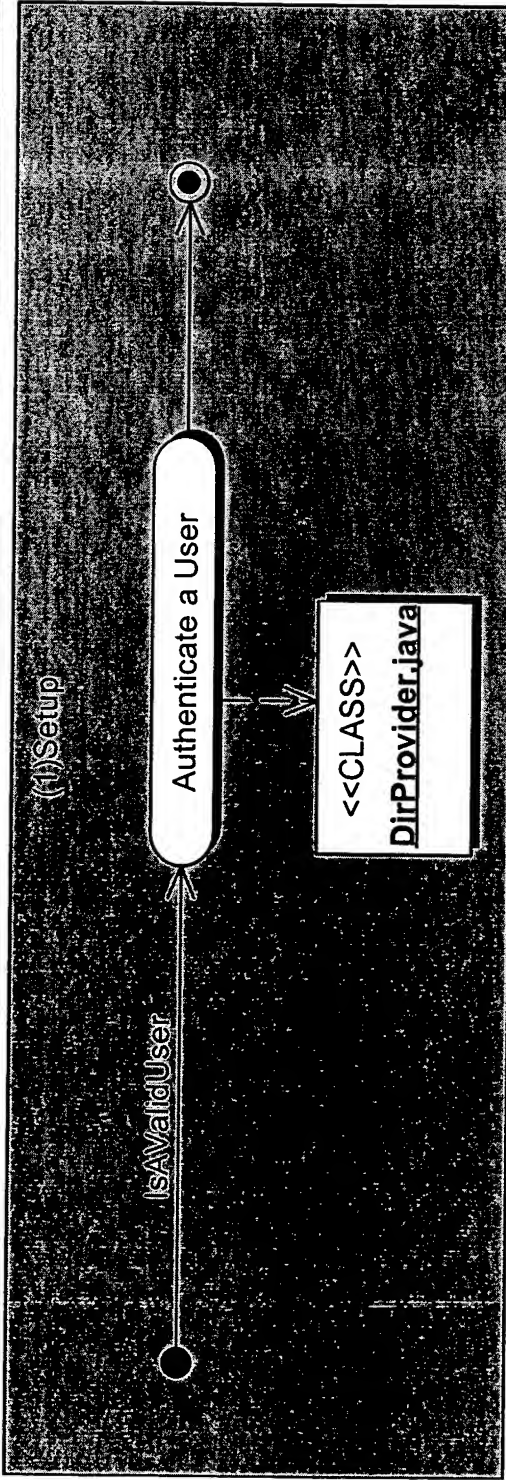
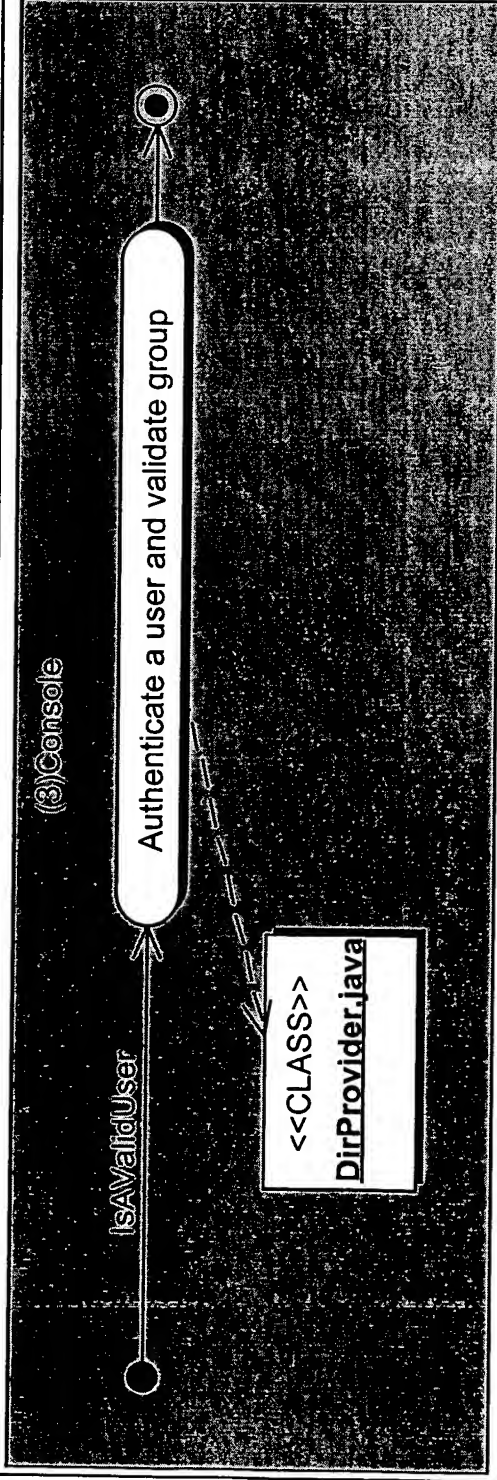


1.1, (4)txSync parameters screen

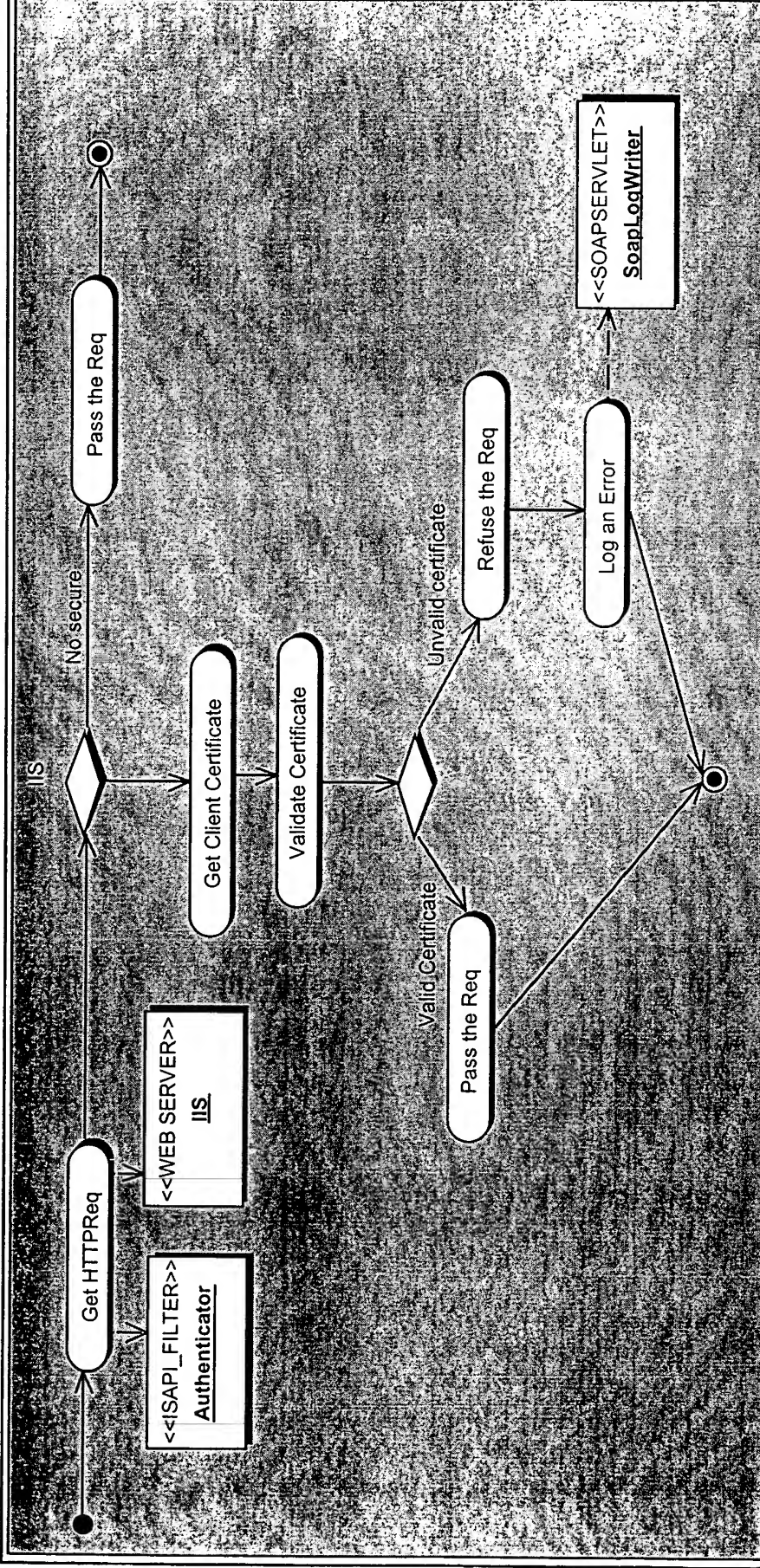


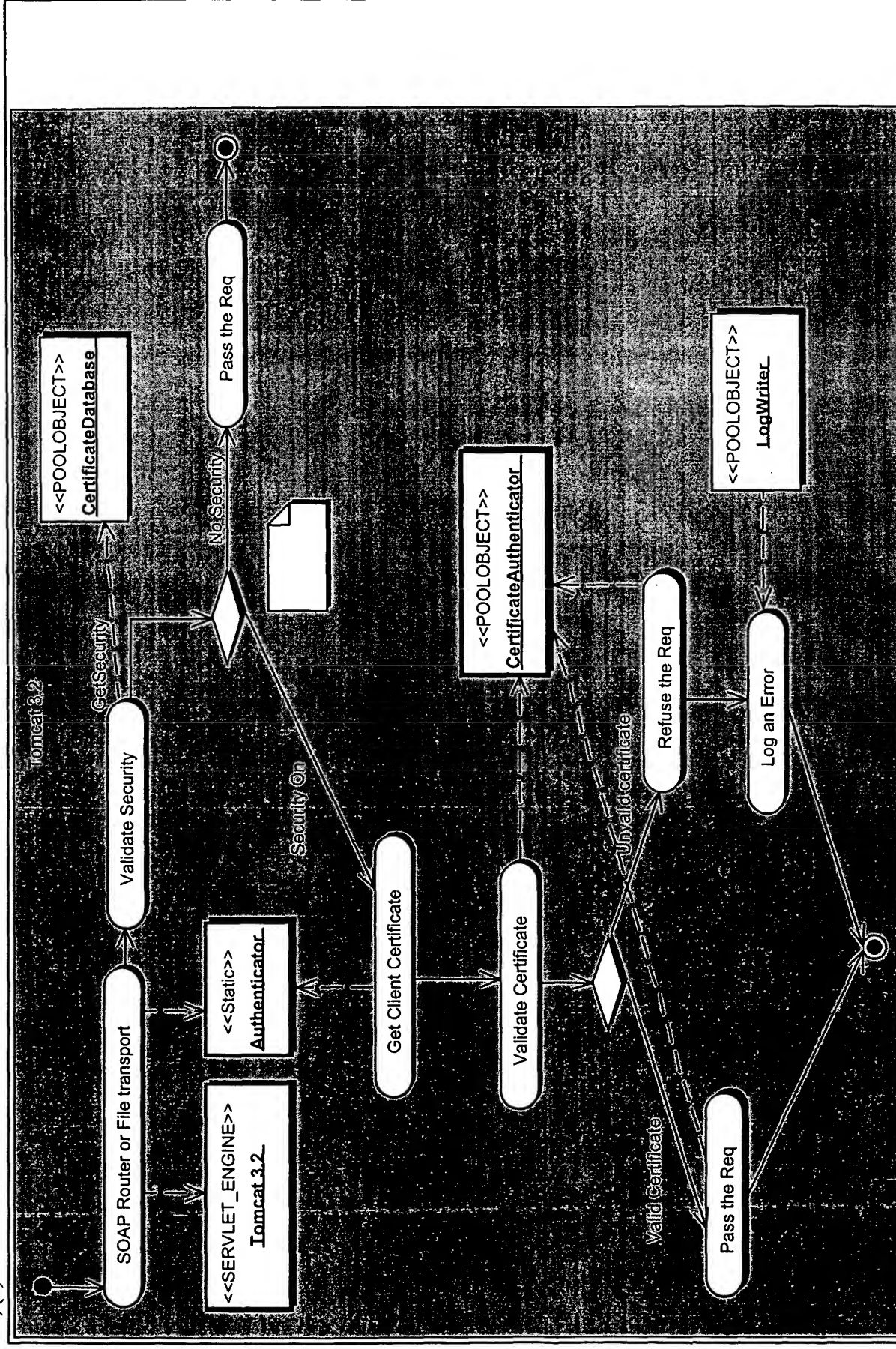
1.1, (4)txSync parameters screen

1.1, (5)Users Authentication

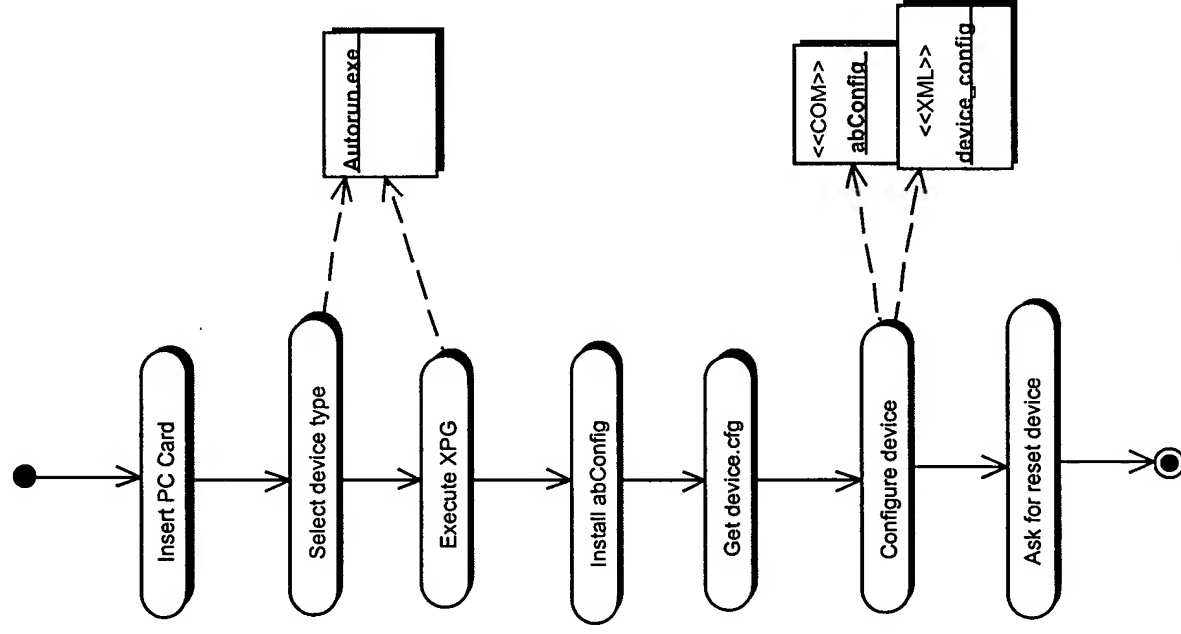


1.1, (5)Users Authentication

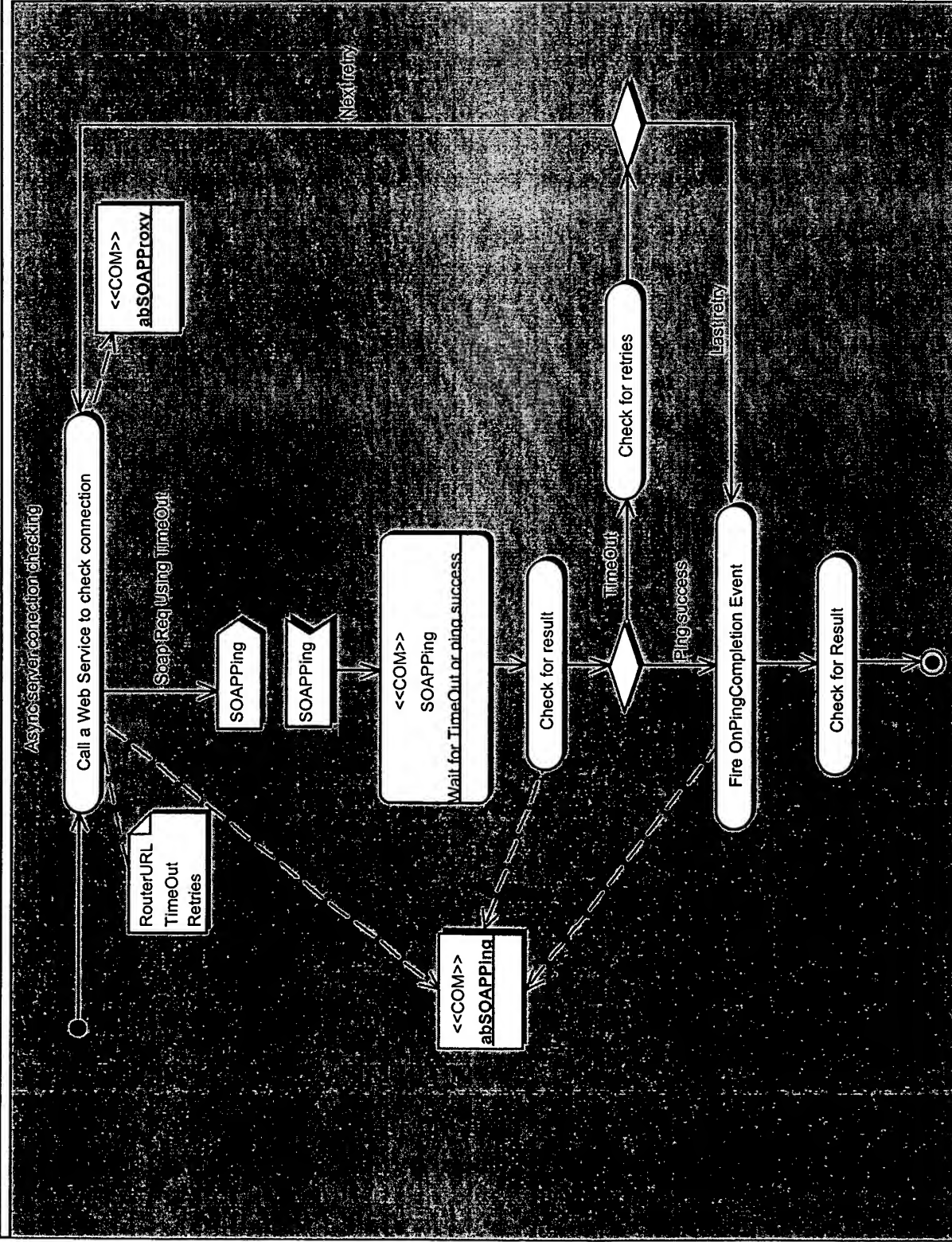




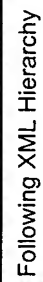
1.1, (7)Auto conection configuration



1.1, (7)Auto conection configuration

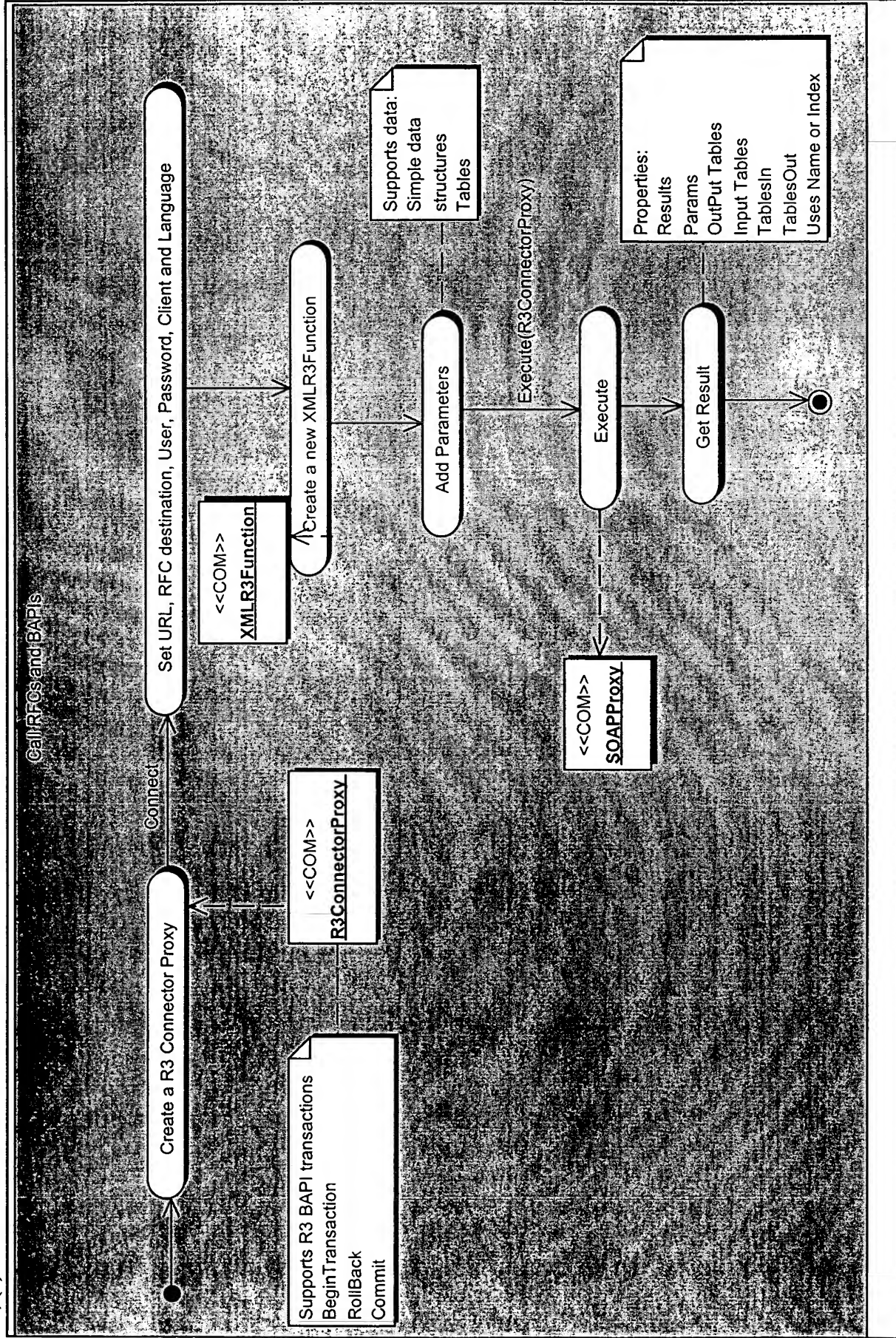


CallIDOCs



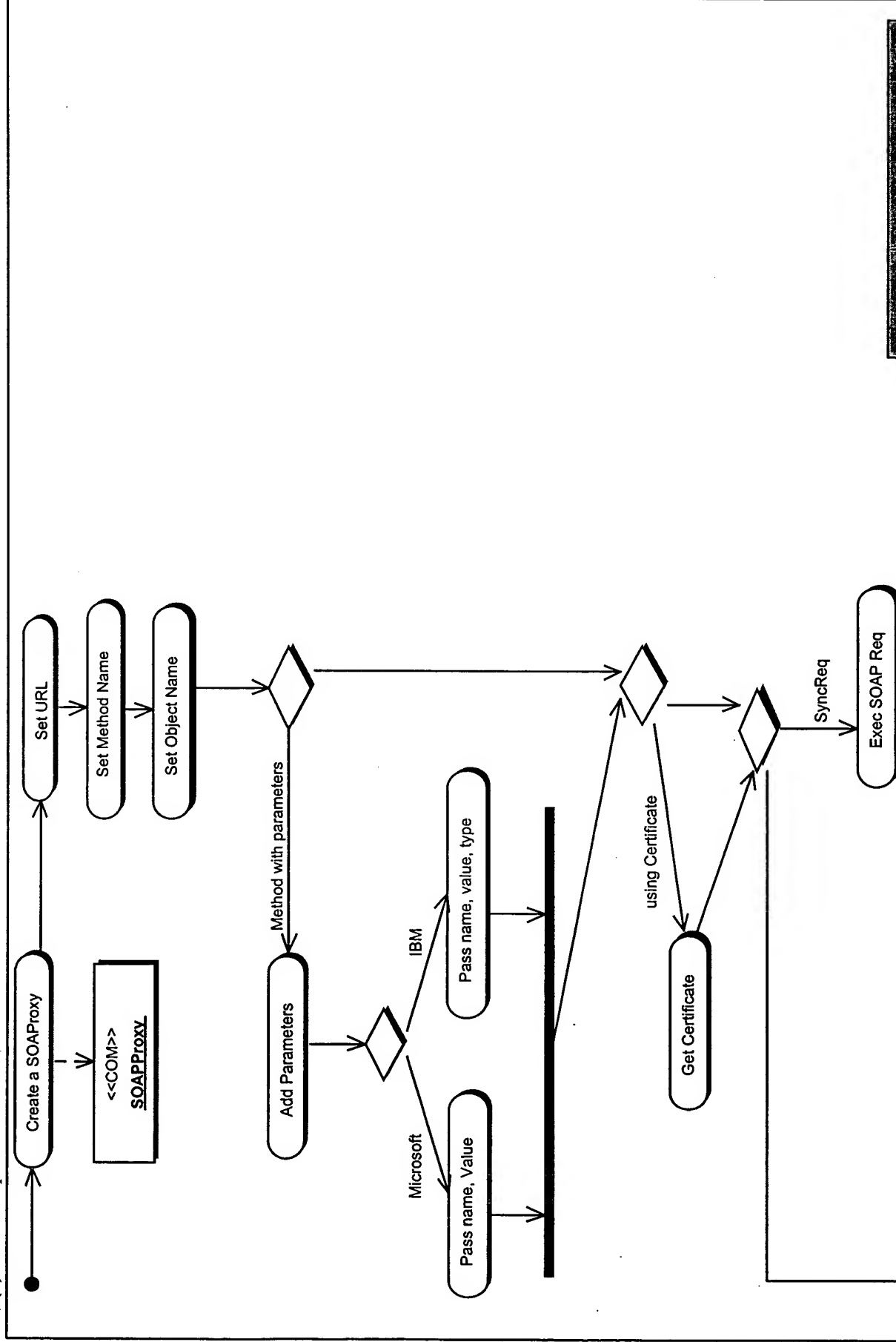
1.1, (7)R3 CP IDOC Post

1.1, (7)R3 CP RFC & BAPI Call



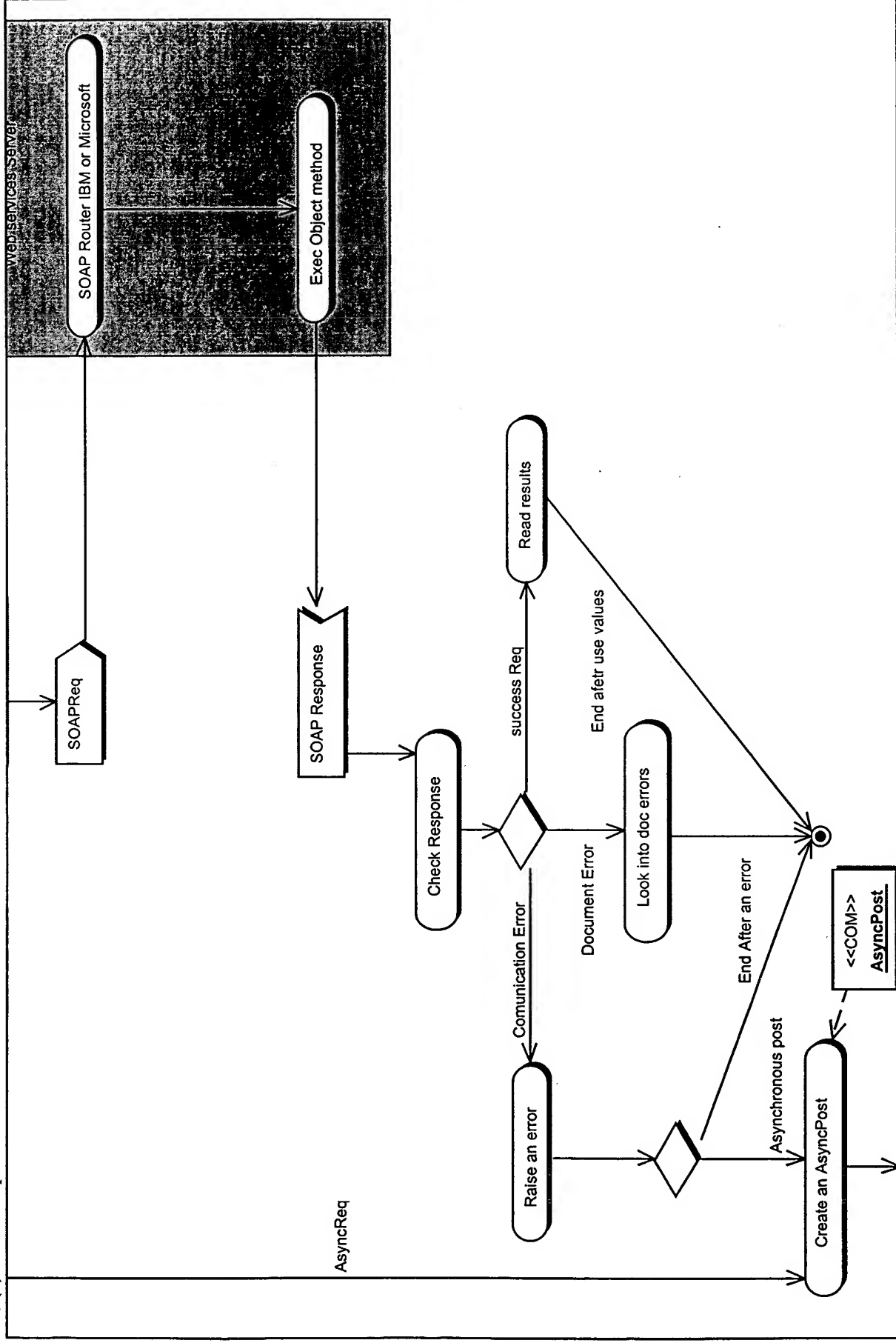
1.1, (7)R3 CP RFC & BAPI Call

1.1.1, (7)SOAP Request



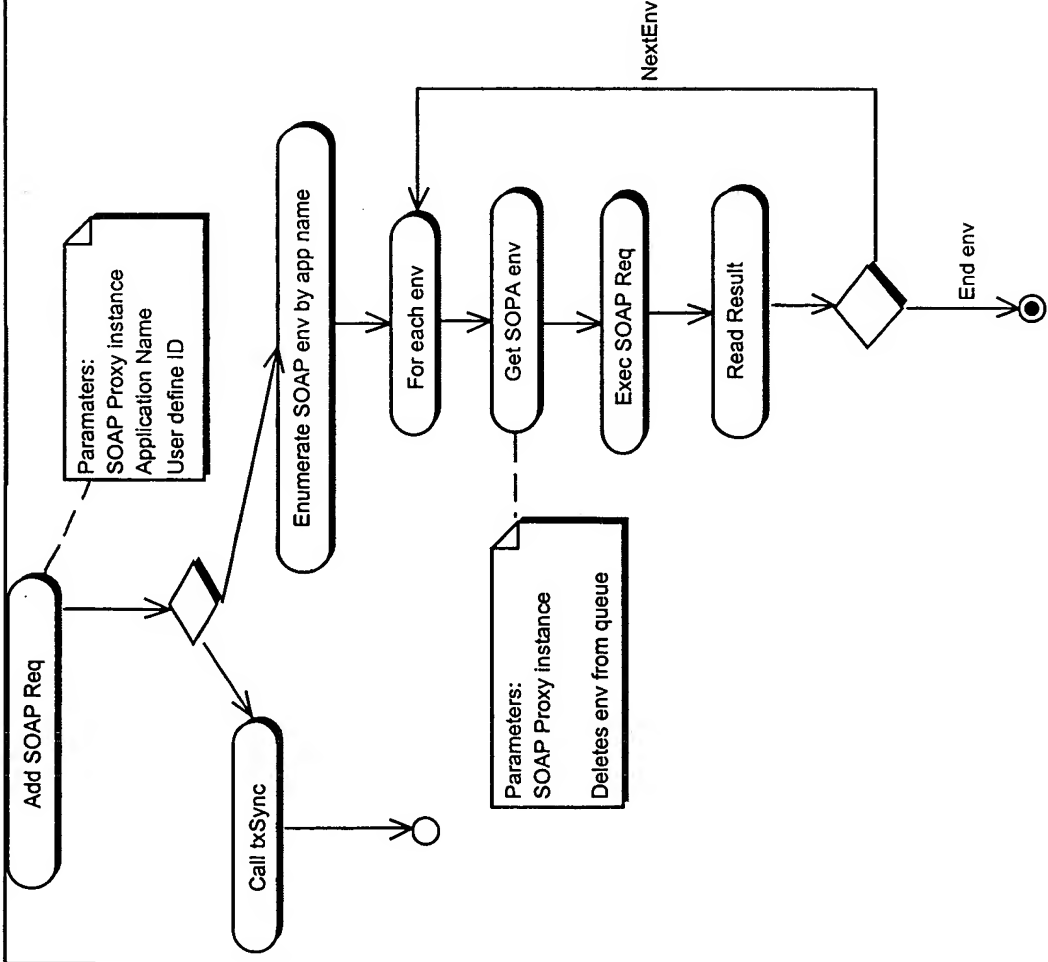
1.1.1, (7)SOAP Request

2.1.1, (7)SOAP Request



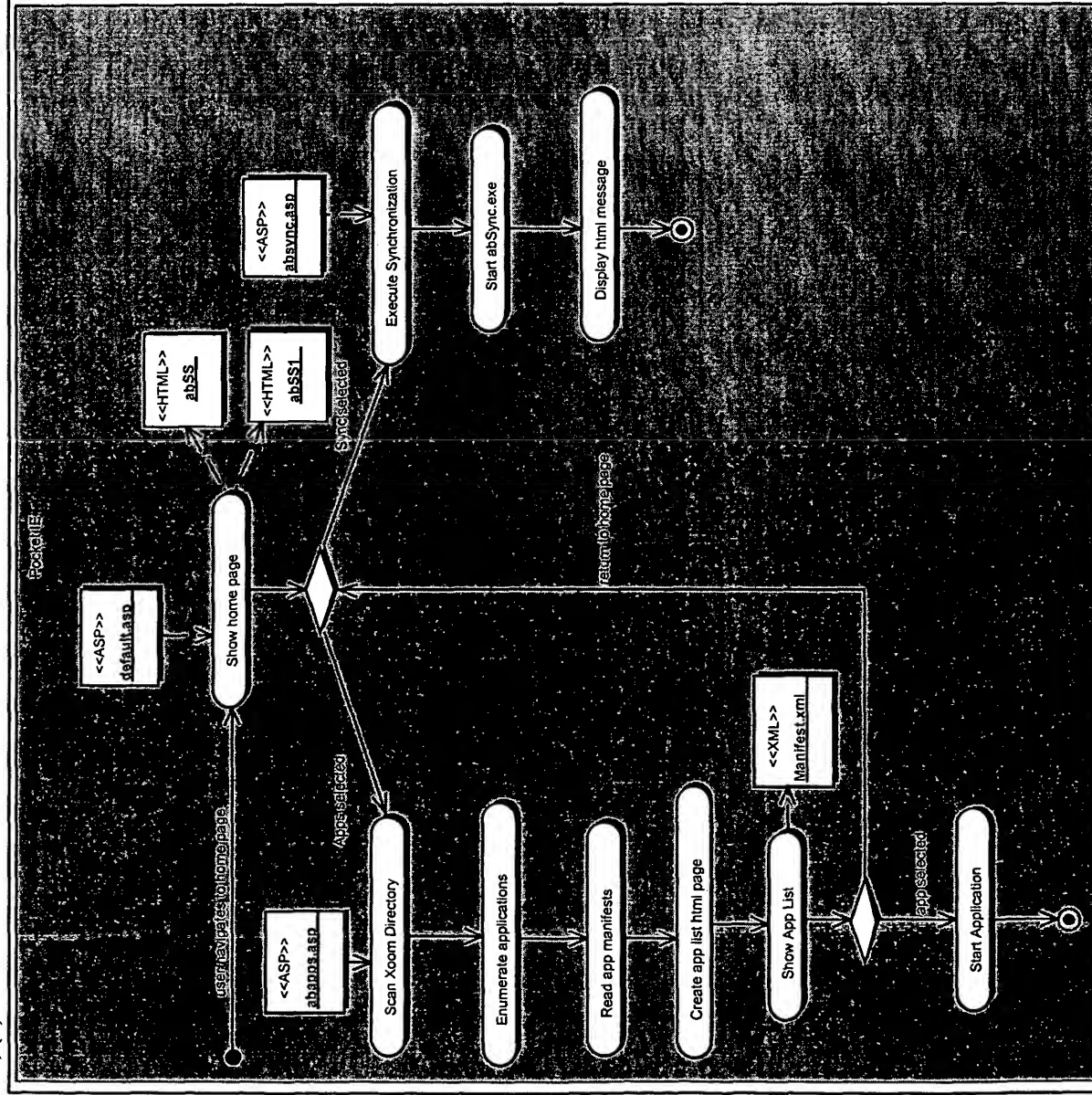
2.1.1, (7)SOAP Request

3.1, (7)SOAP Request

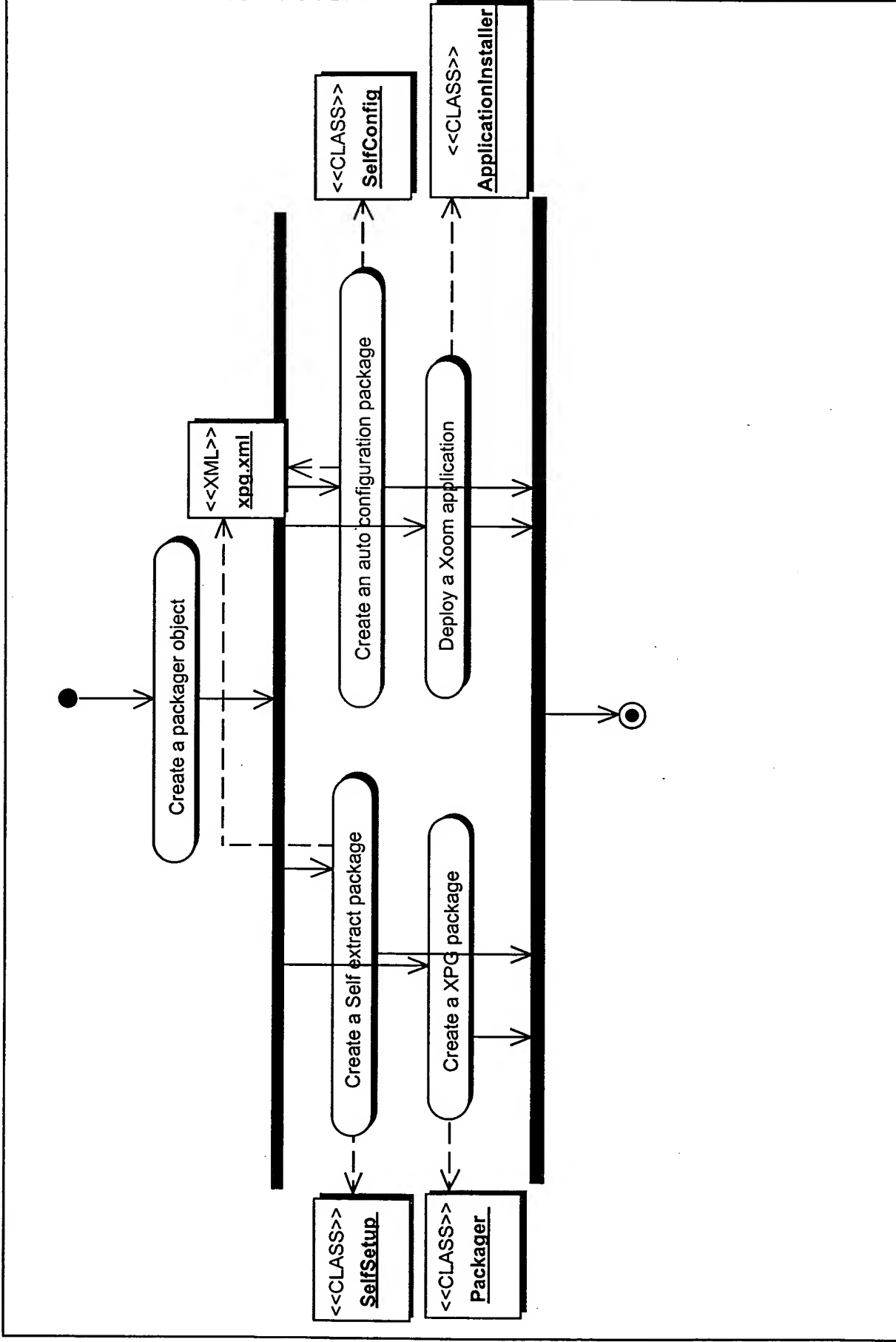


3.1, (7)SOAP Request

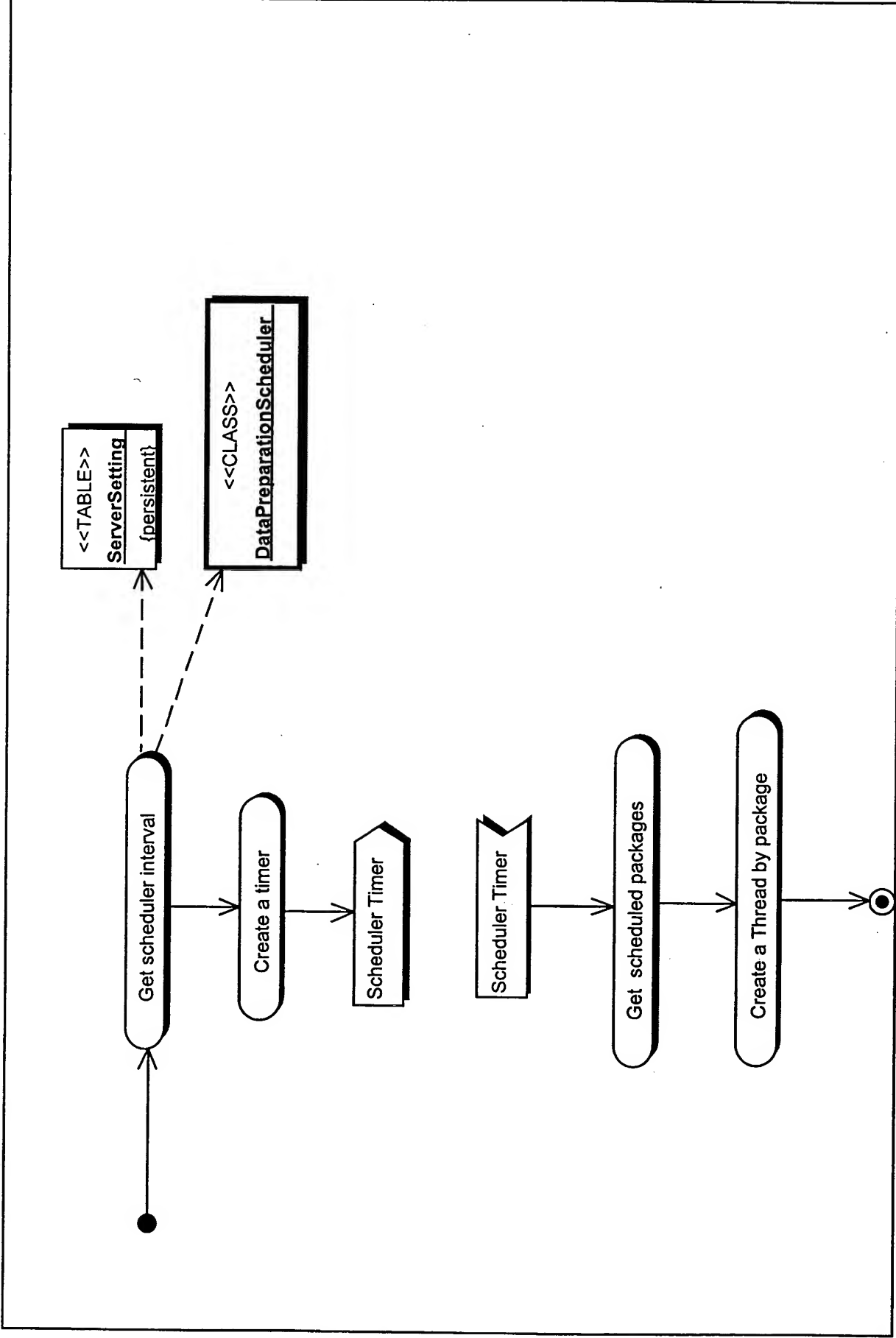
1.1, (7)Xoom Shell



1.1, (7)Xoom Shell

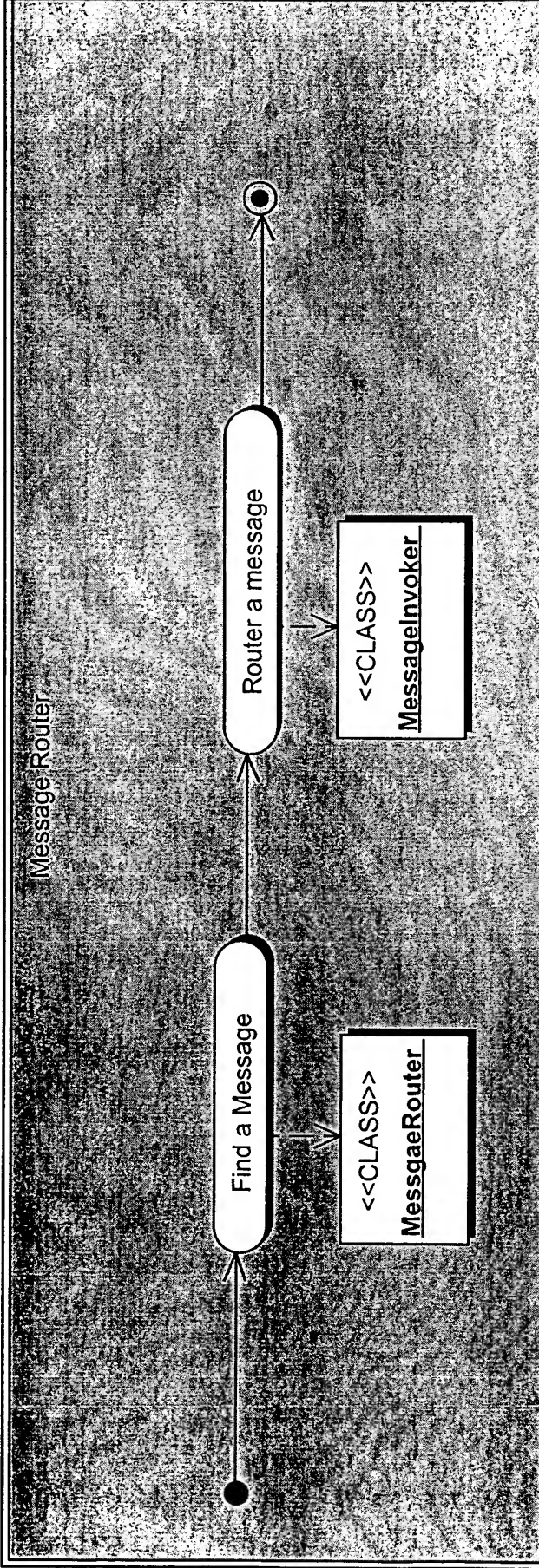


1.1, (9)Data preparation by scheduler



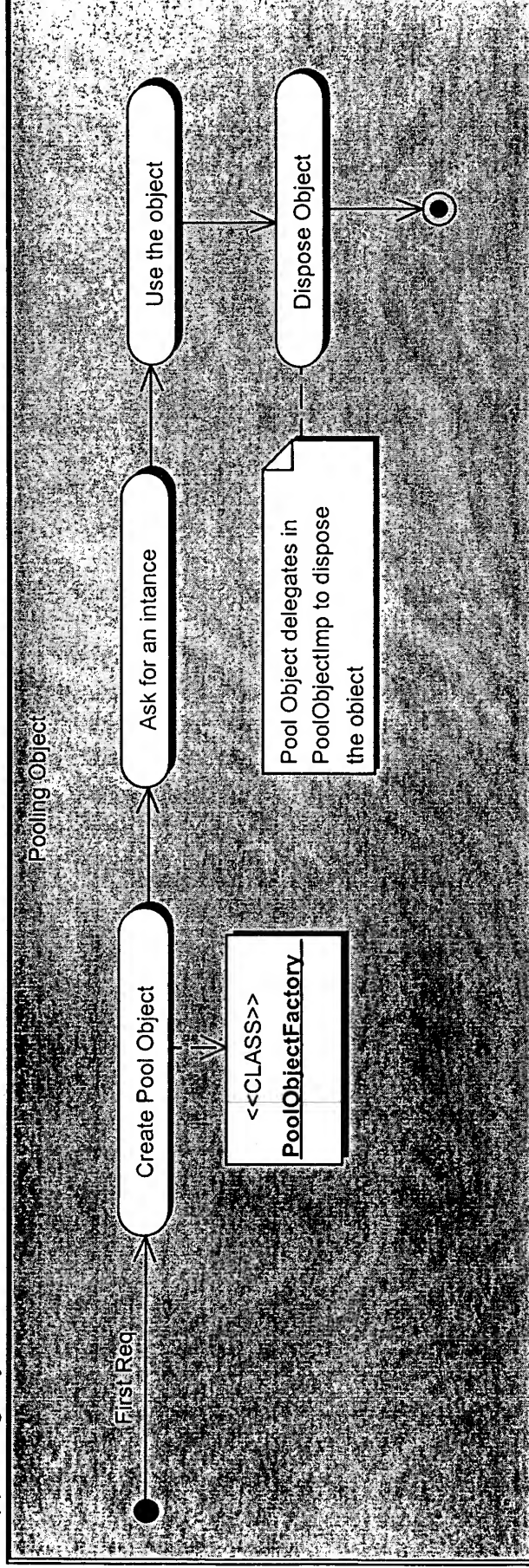
1.1, (9)Data preparation by scheduler

1.1, (9)Message router



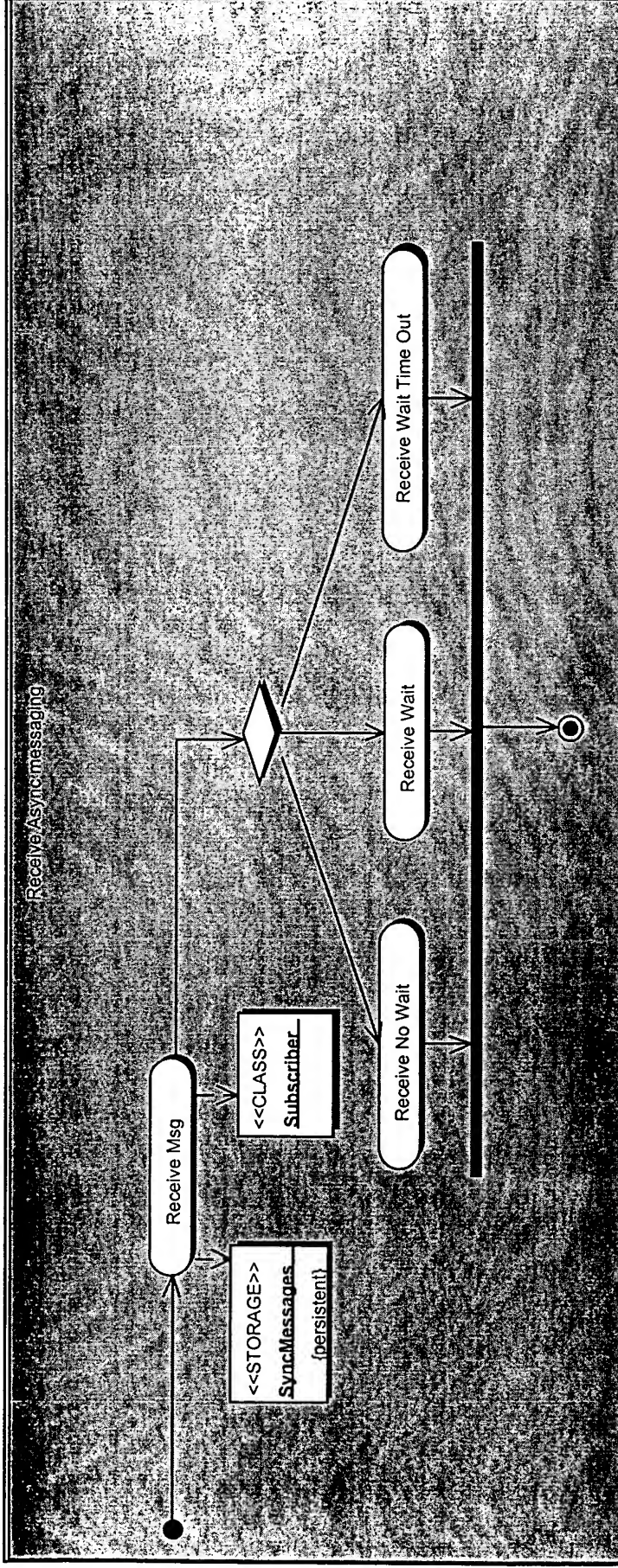
1.1, (9)Message router

1.1, (9) Pooling Object



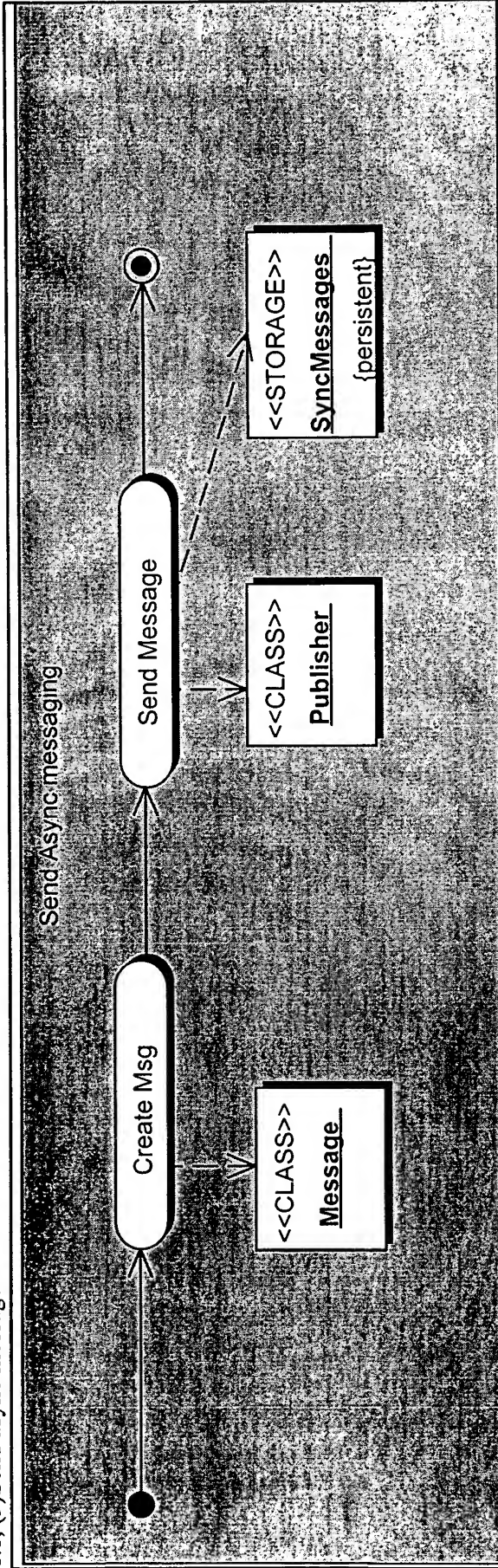
1.1, (9) Pooling Object

1.1, (9)Receive async messaging



1.1, (9)Receive async messaging

1.1, (9)Send async message

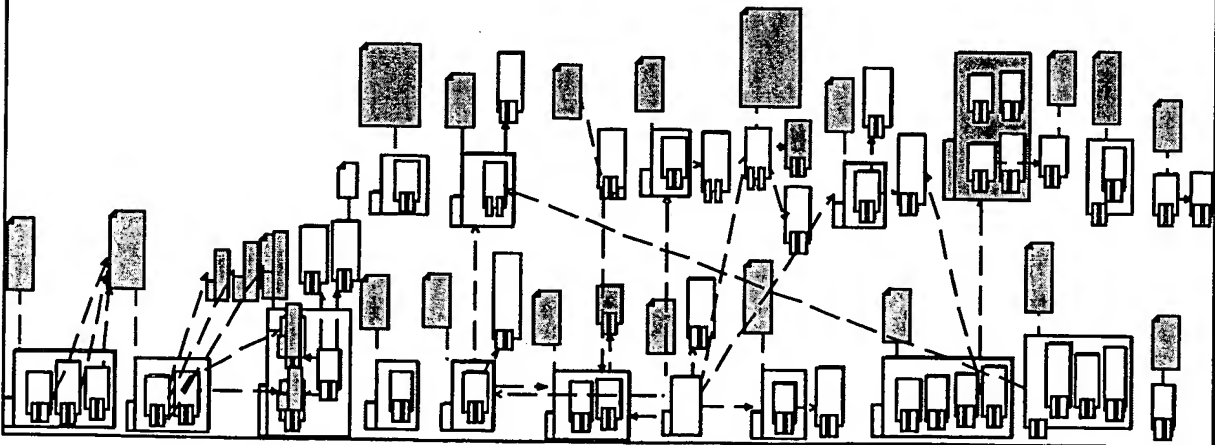


1.1, (9)Send async message

Resp:

OS support: NT, 2000, Linux, Unix

1.1, (7)CE Framework



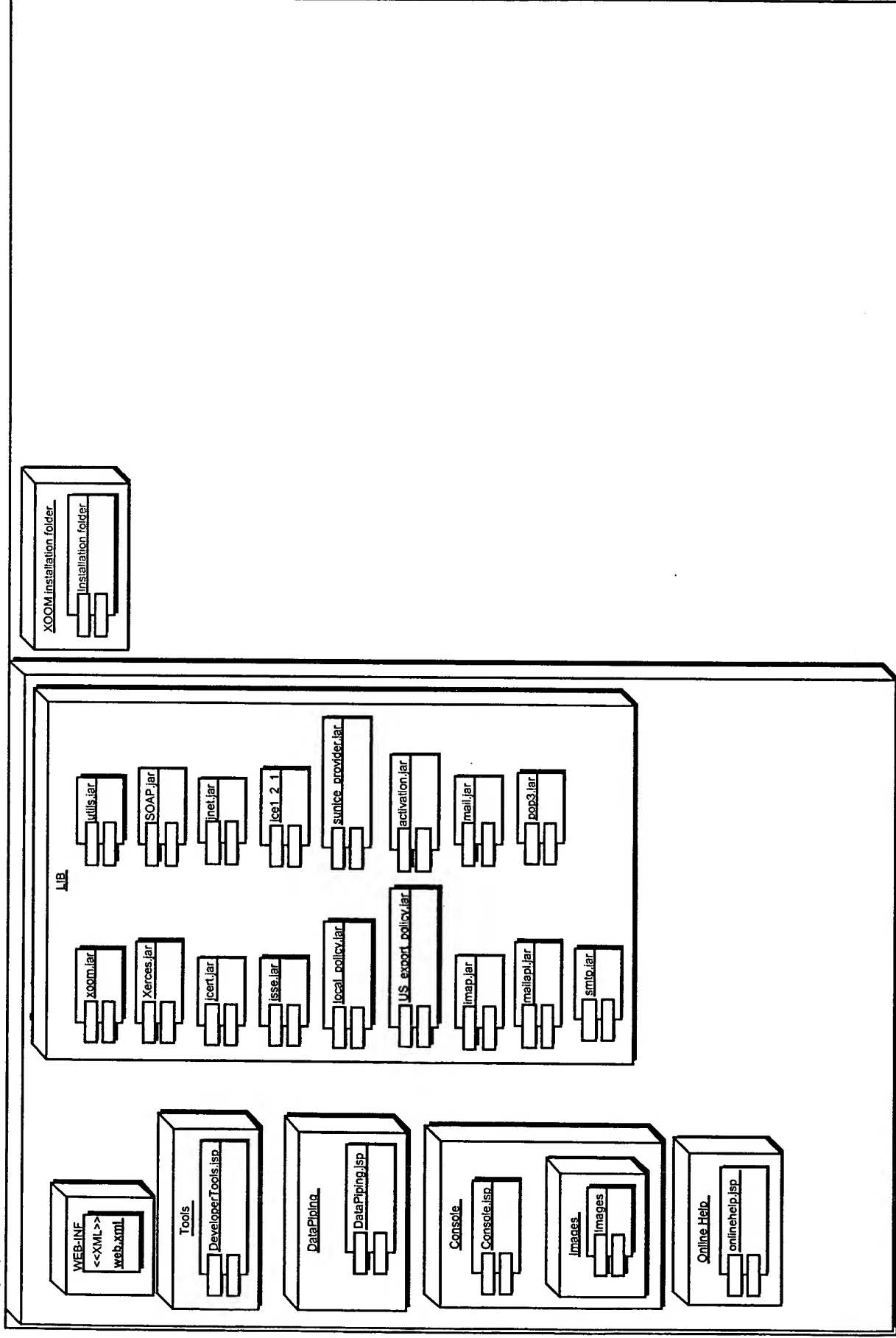
1.1, (7)CE Framework

Resp:

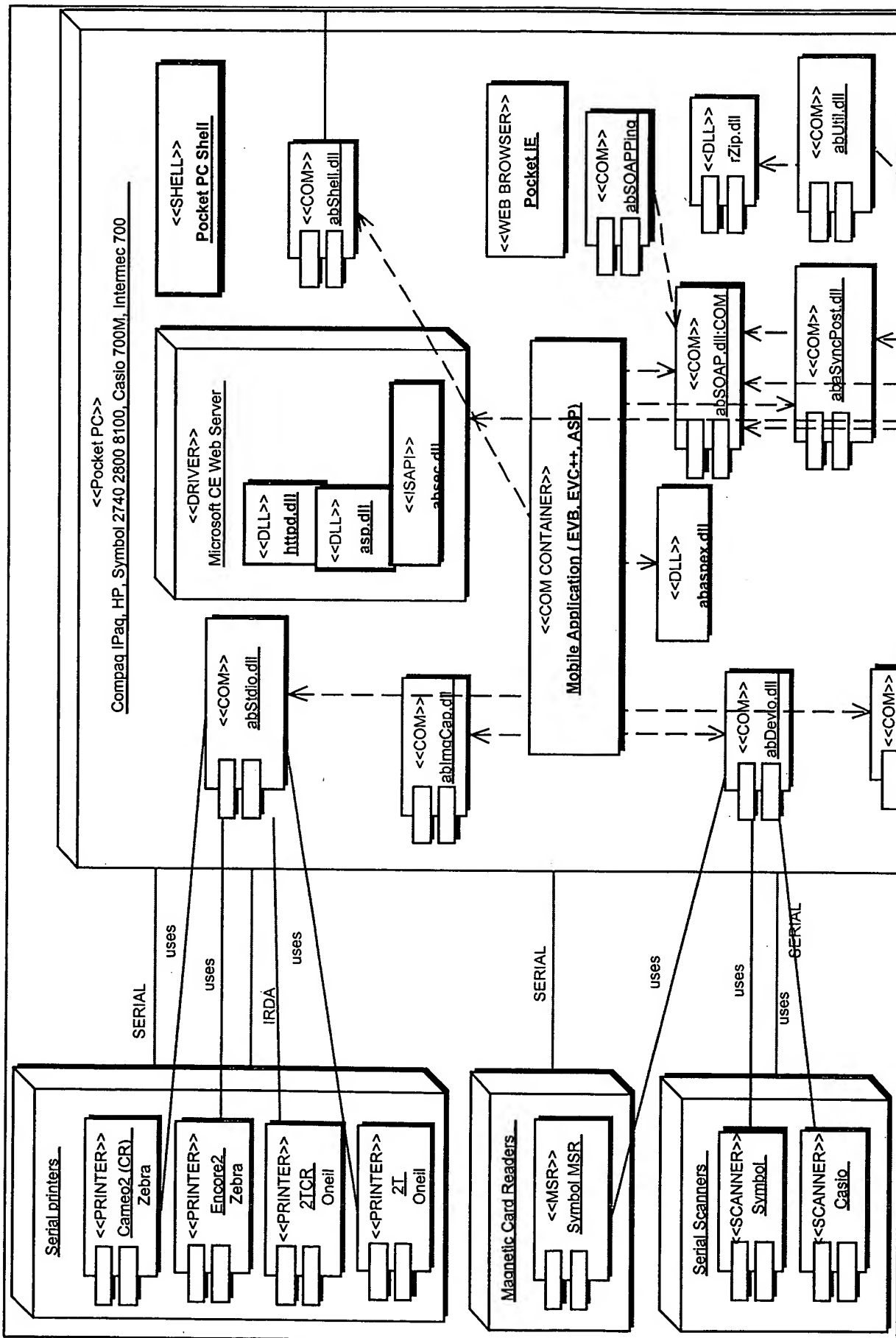
OS support: NT, 2000, Linux, Unix

Resp:

OS support: NT, 2000, Linux, Unix

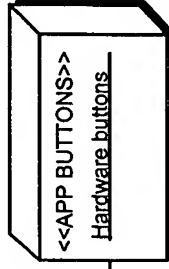


1.1, (7)CE Framework



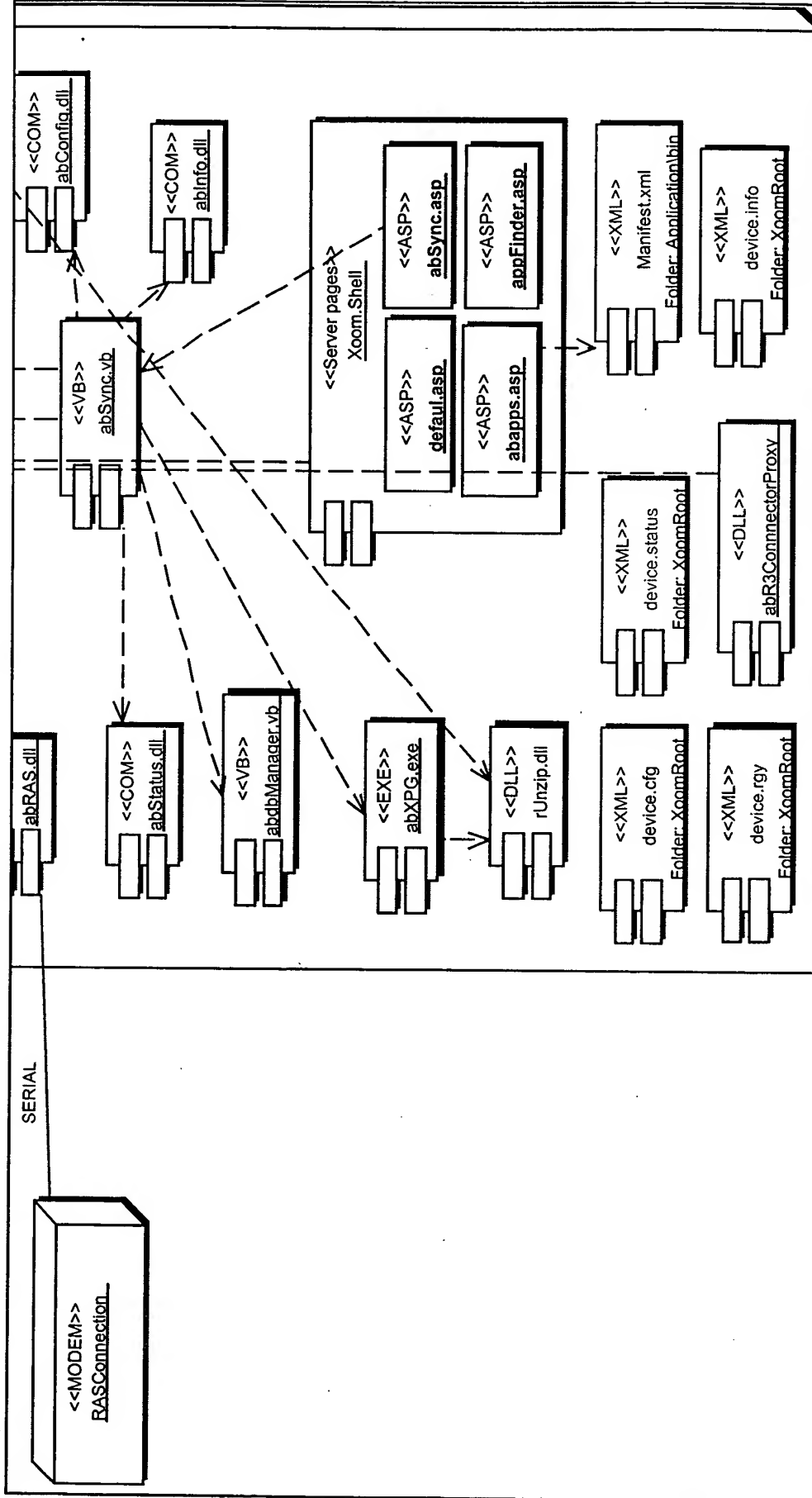
1.1, (7)CE Framework

1.2, (7)CE Framework



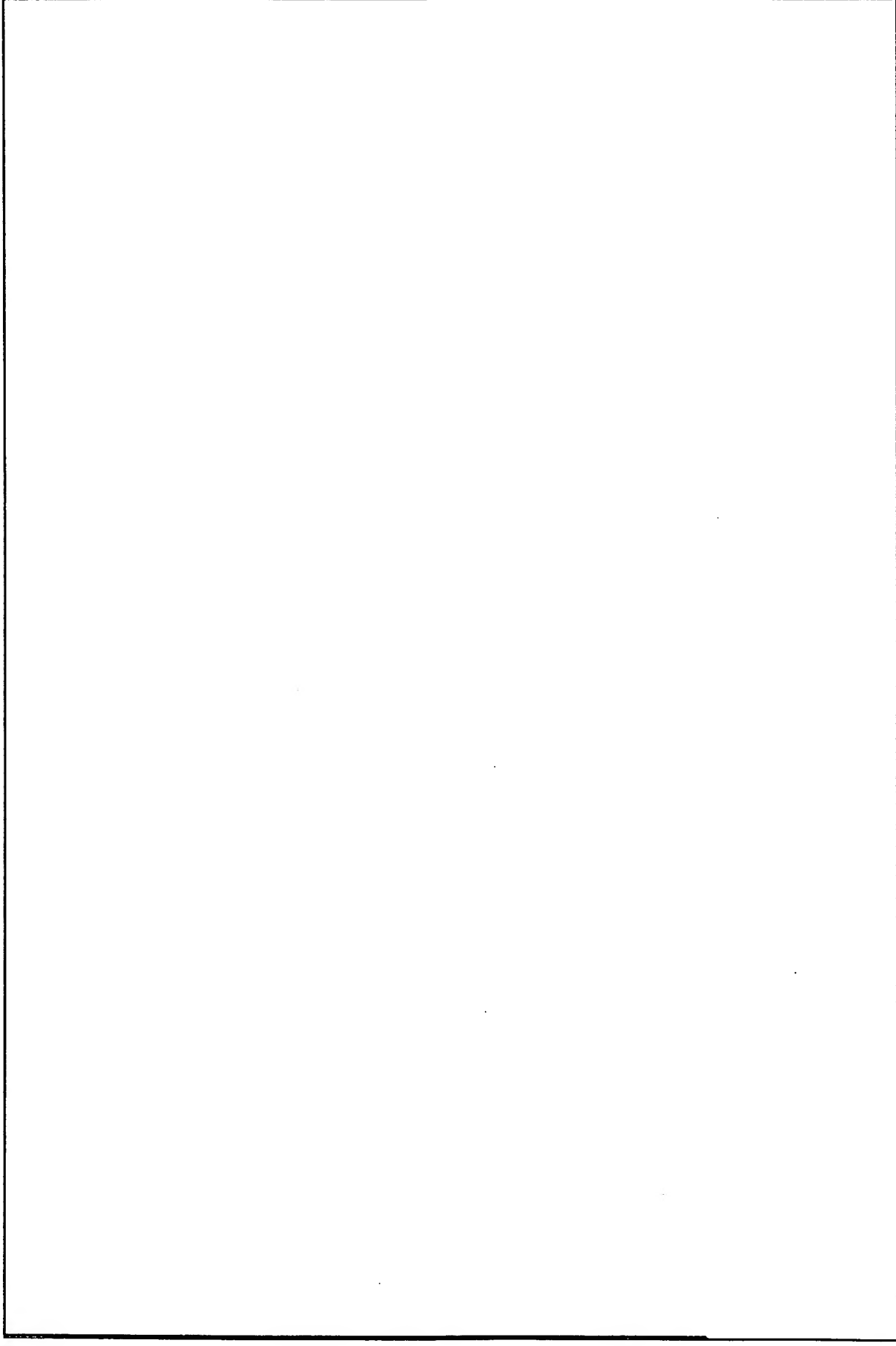
1.2, (7)CE Framework

2.1, (7)CE Framework



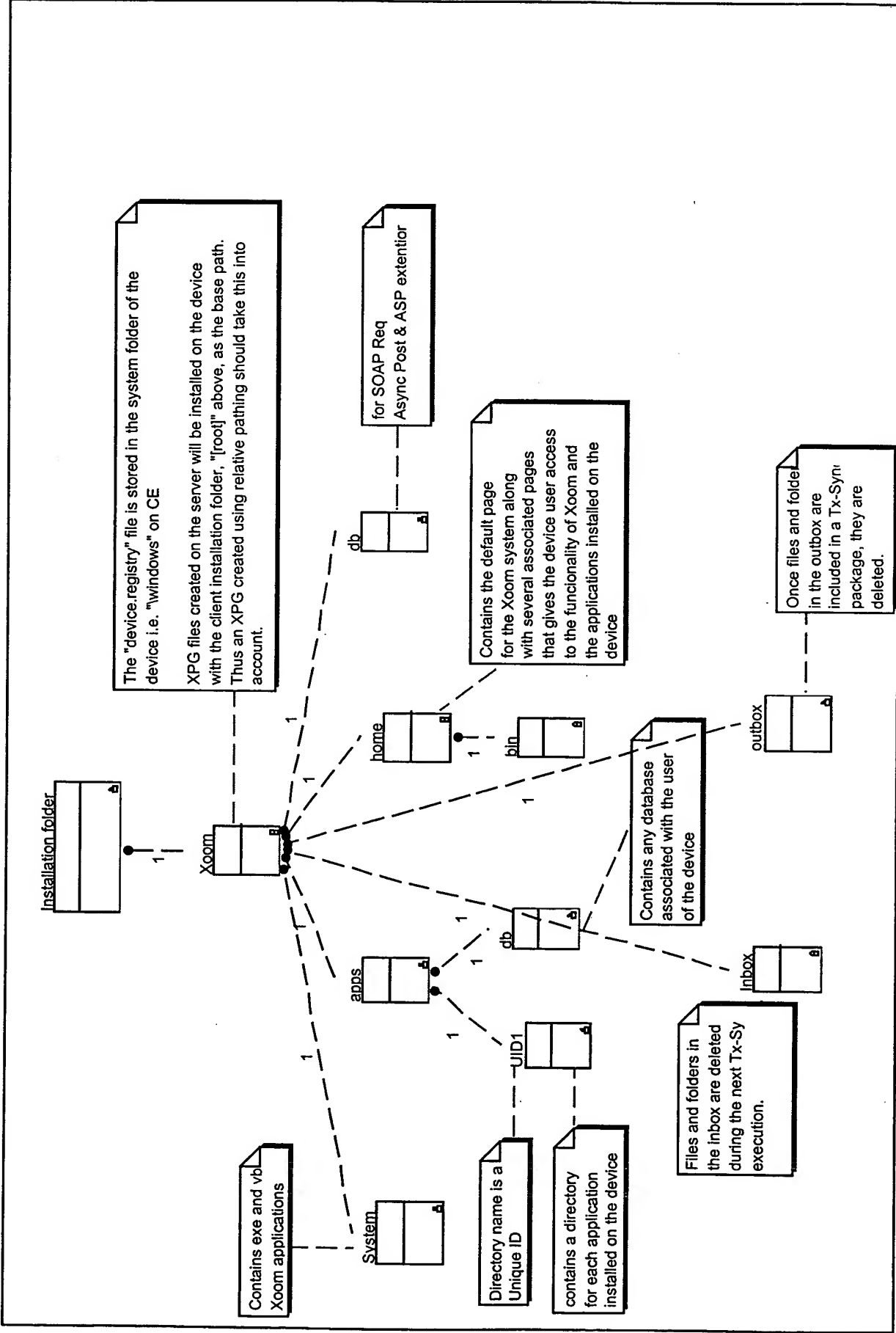
2.1, (7)CE Framework

2.2, (7)CE Framework



2.2, (7)CE Framework

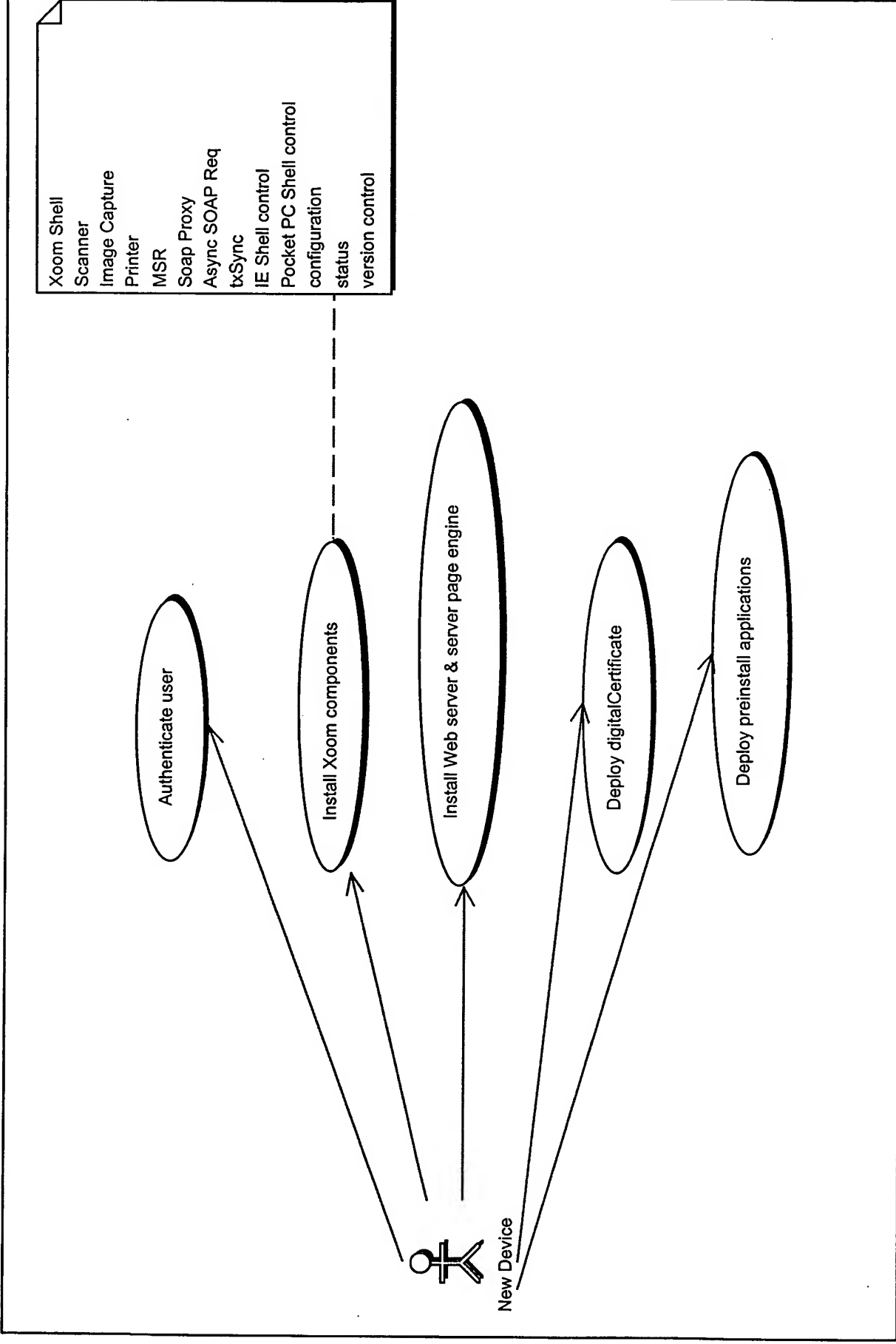
1.1, Client directory structure



1.1, Client directory structure

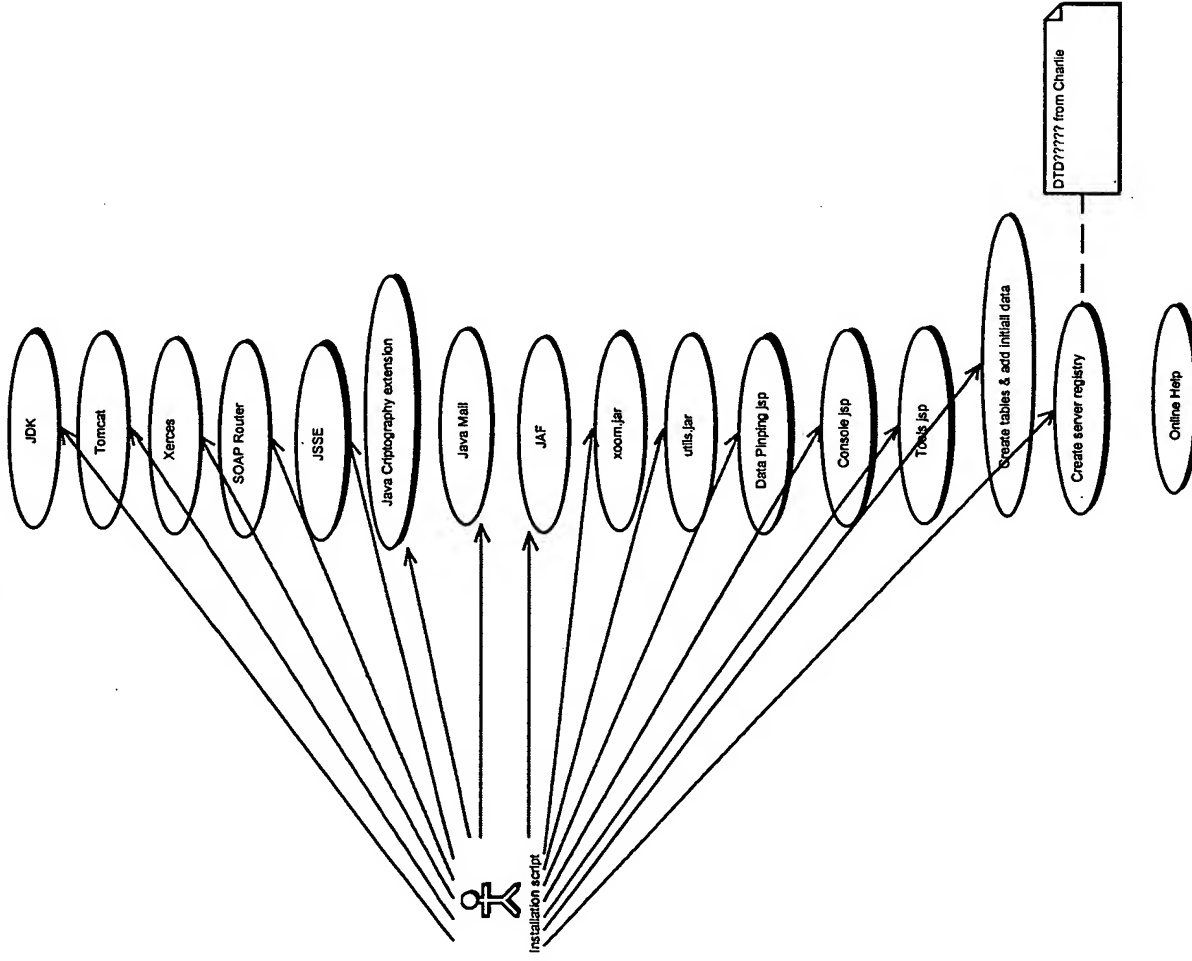
1.1, Server directory structure

1.1.1, (1)Setup

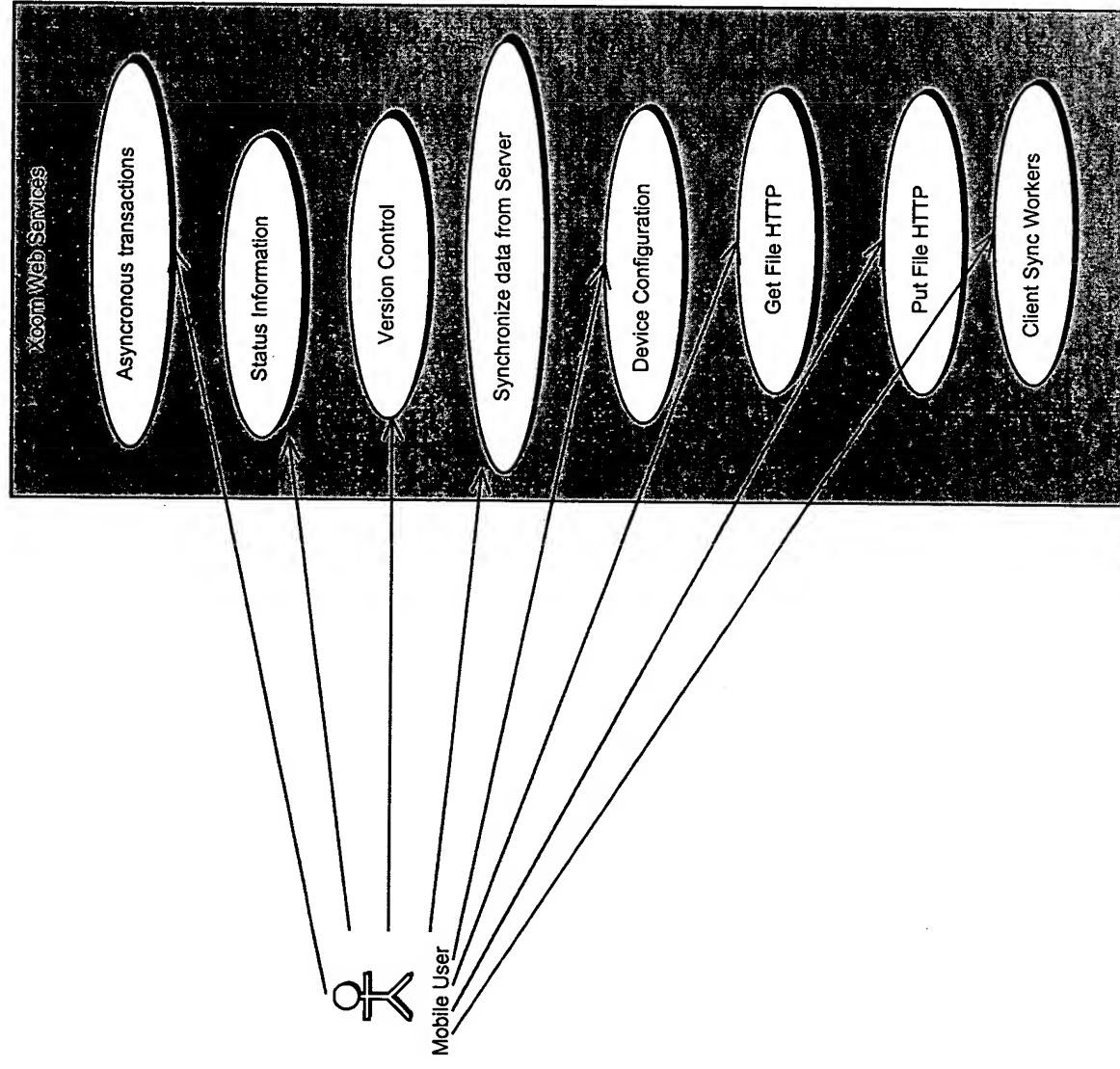


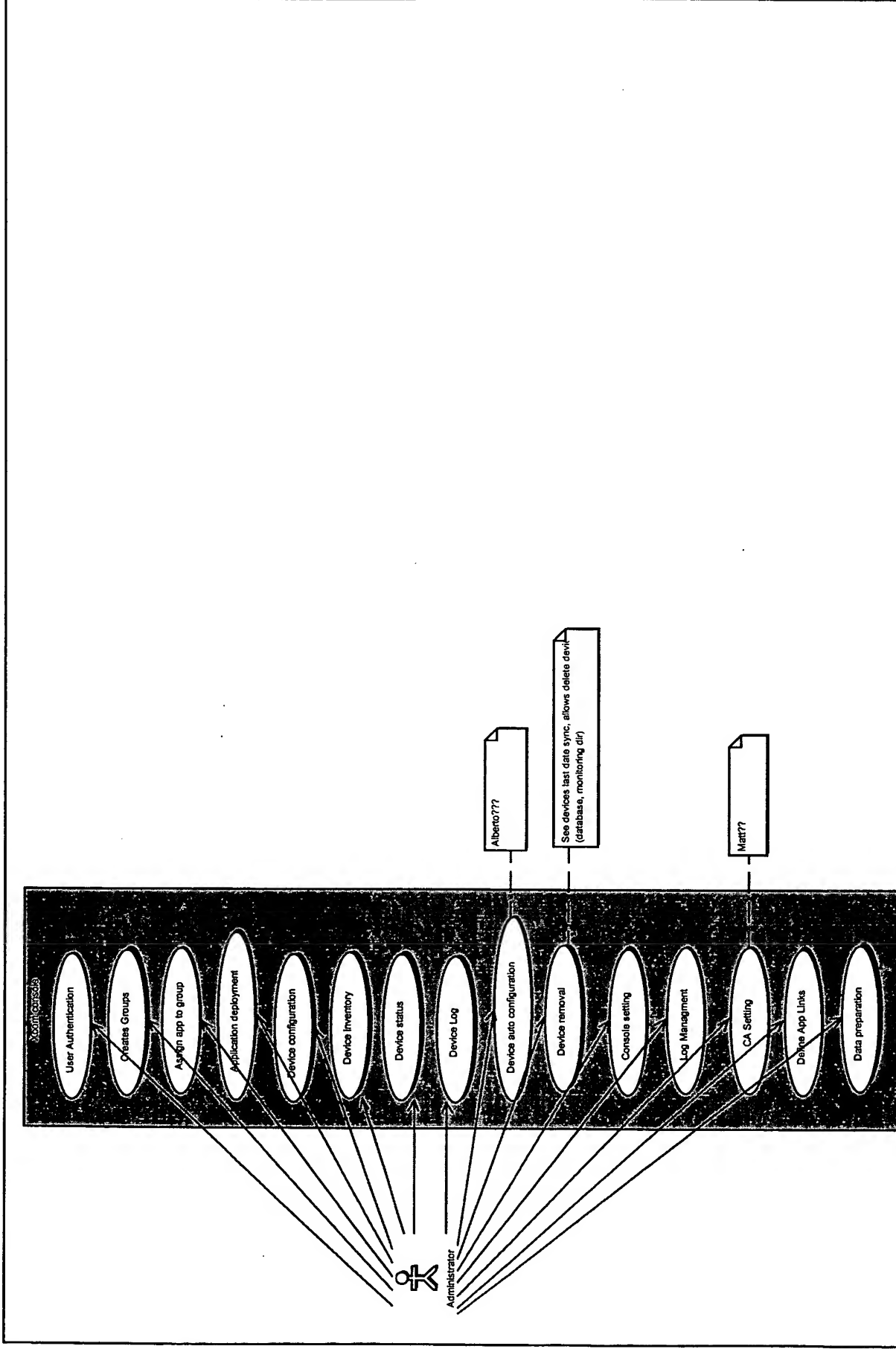
1.1.1, (1)Setup

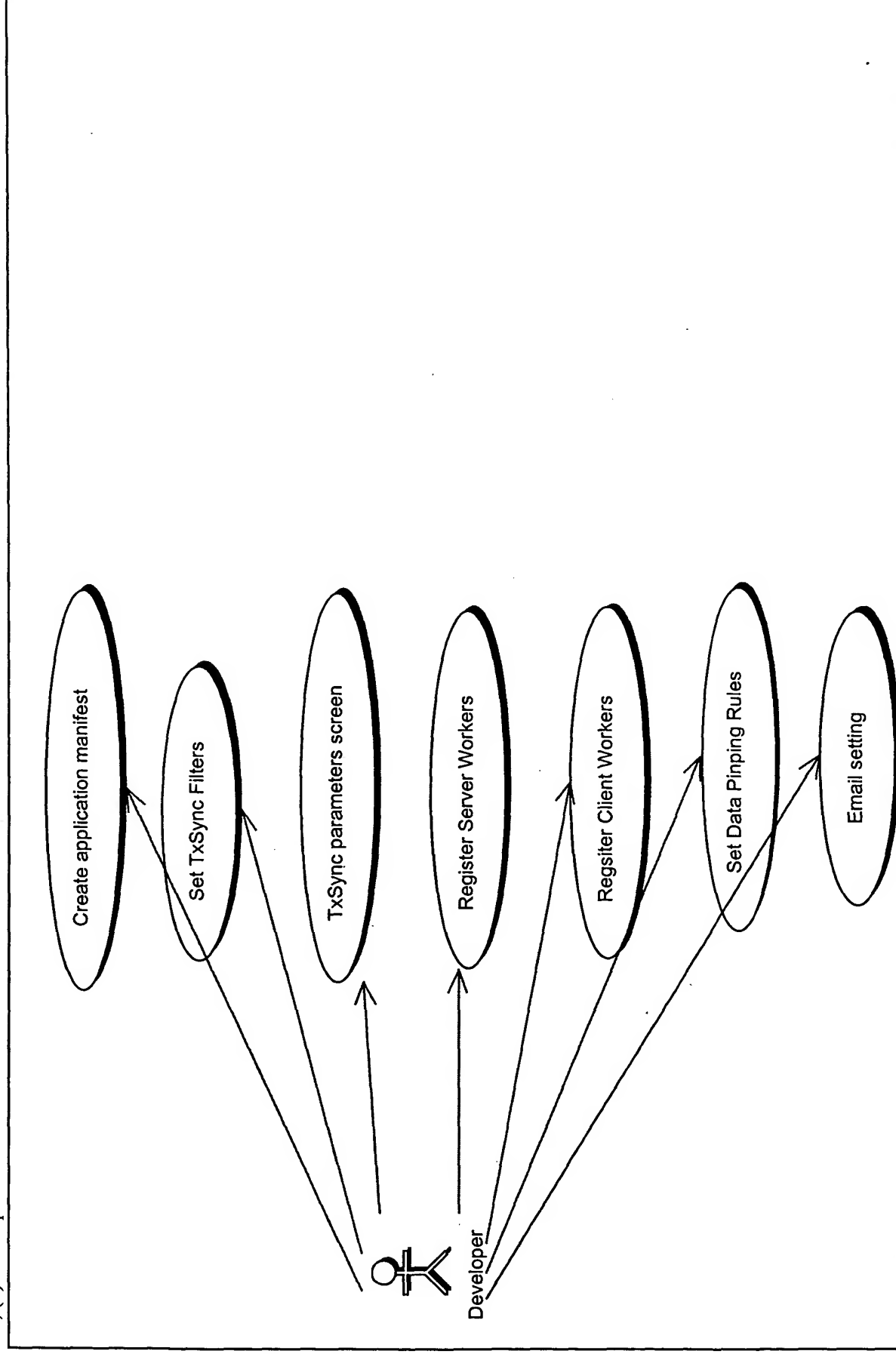
1.1, (10)Xoom Services Installation

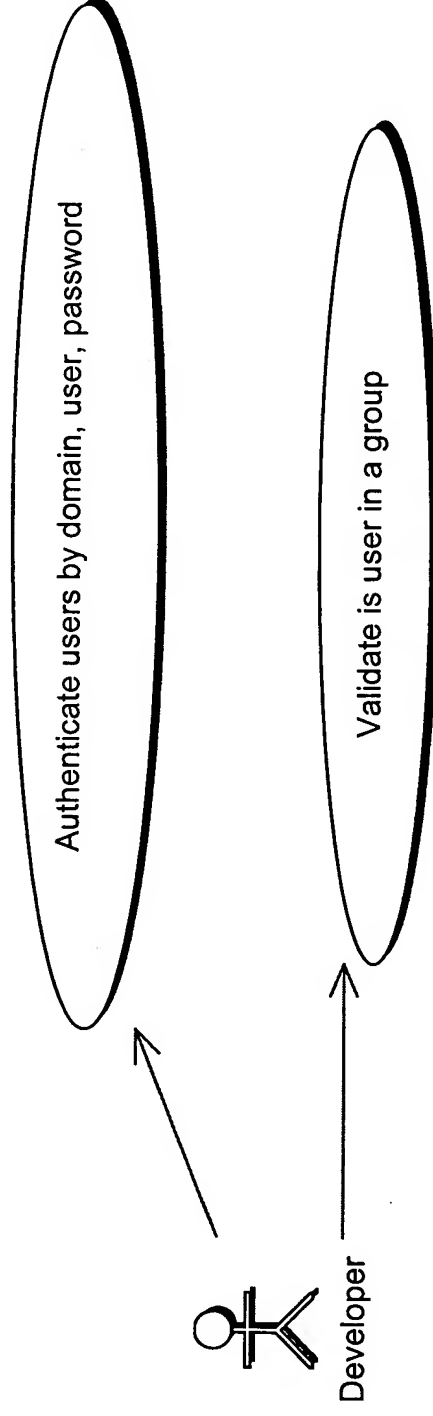


1.1, (10)Xoom Services Installation

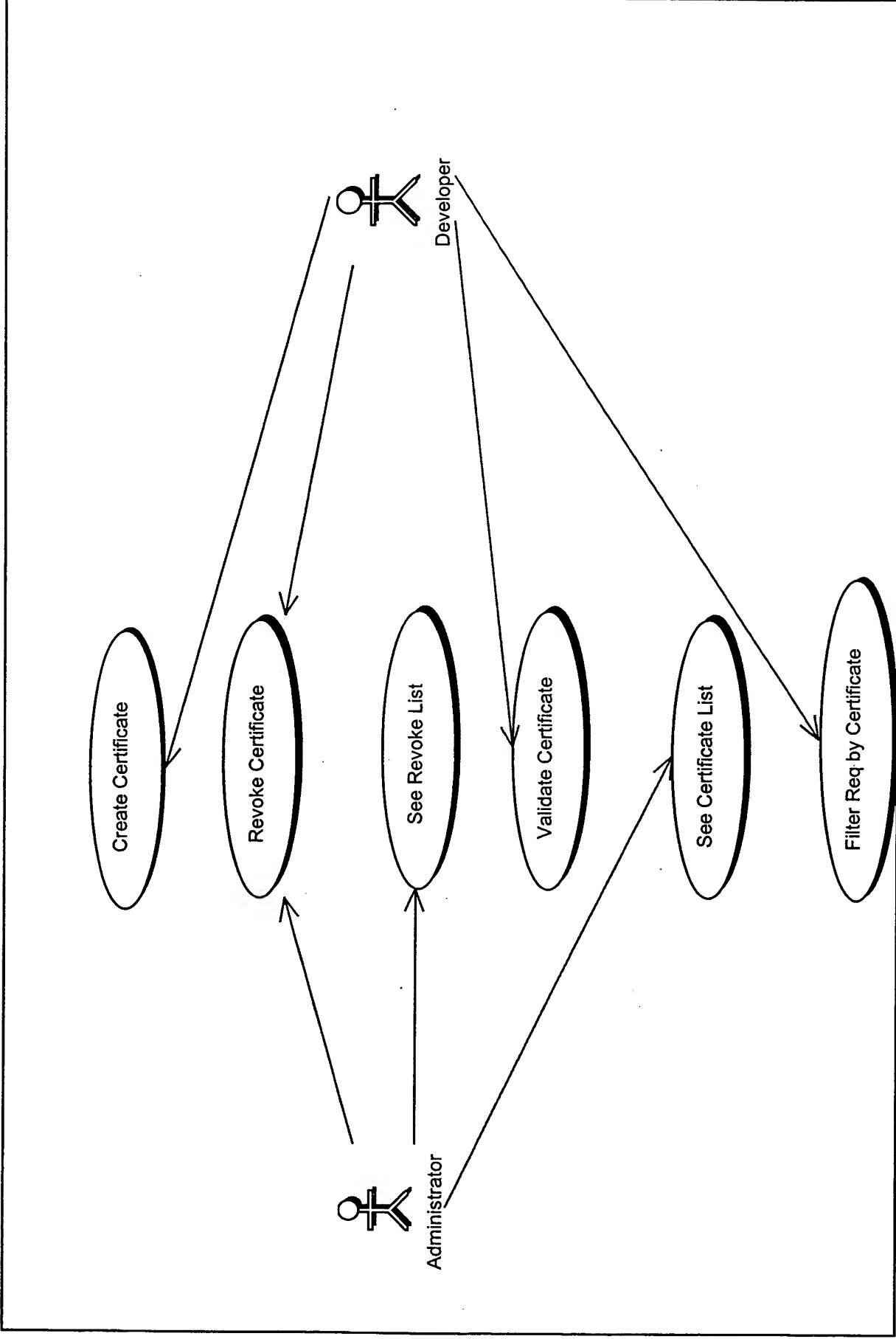






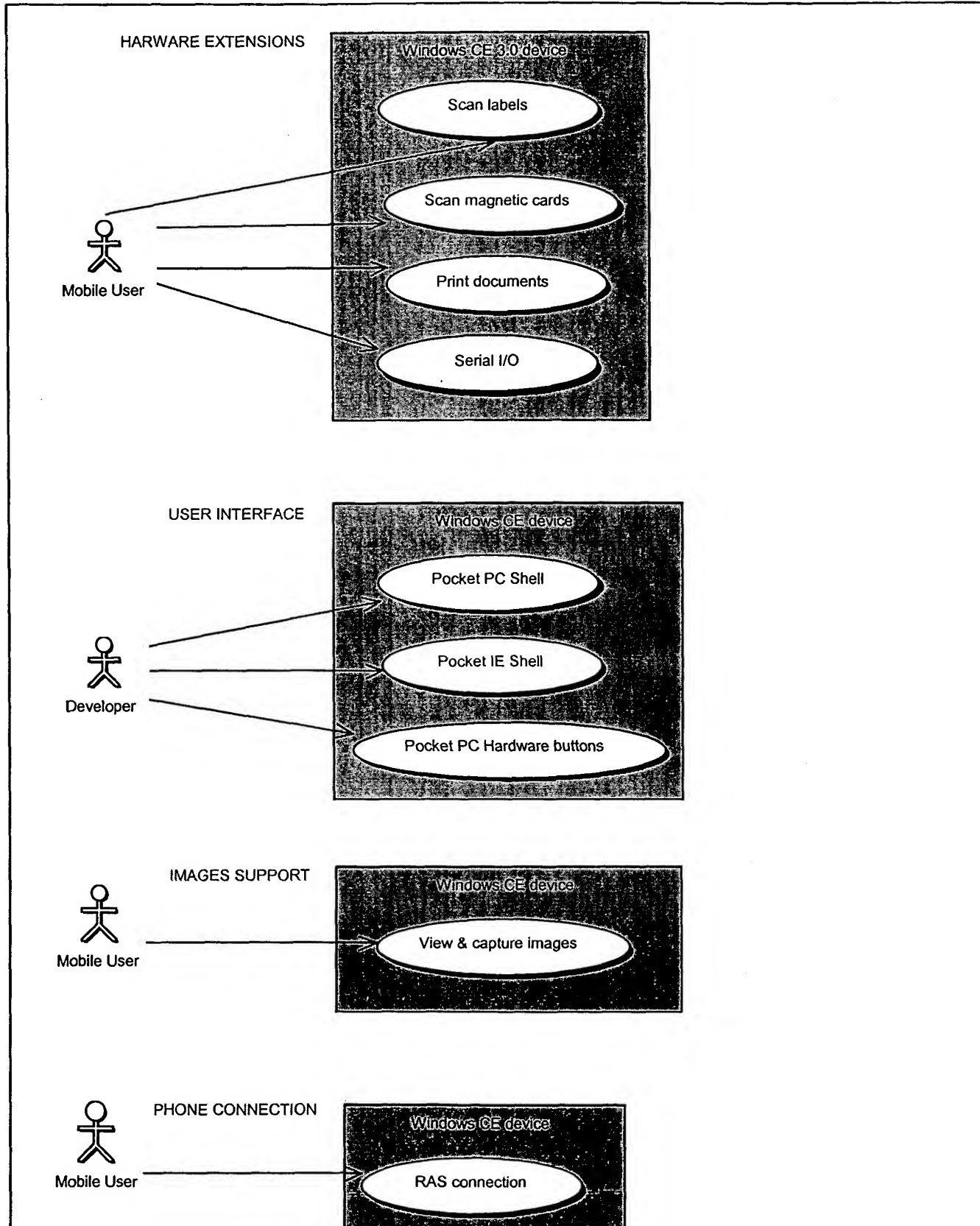


1.1., (6)SSL Client Authentication



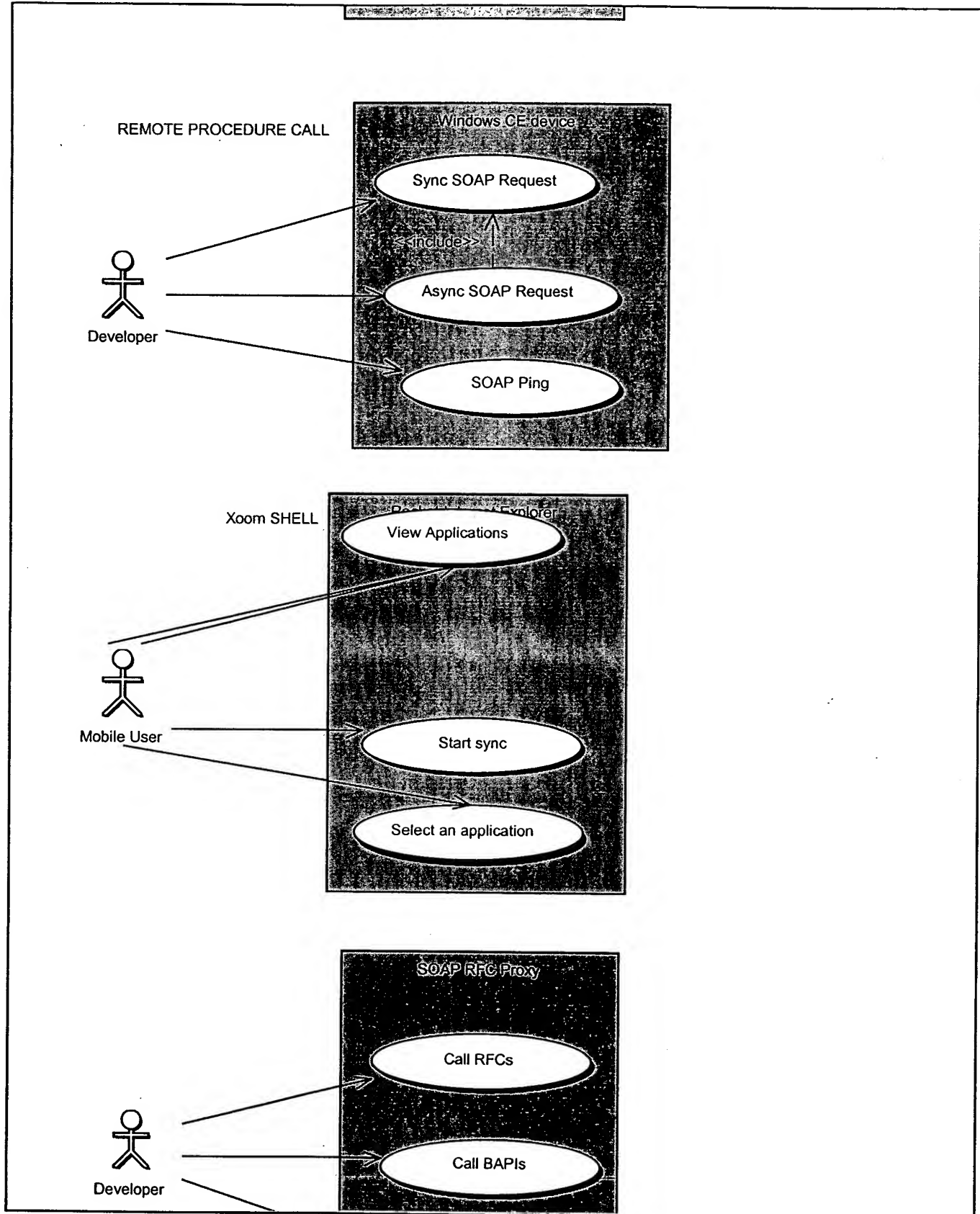
1.1., (6)SSL Client Authentication

1.1, (7)CE Framework



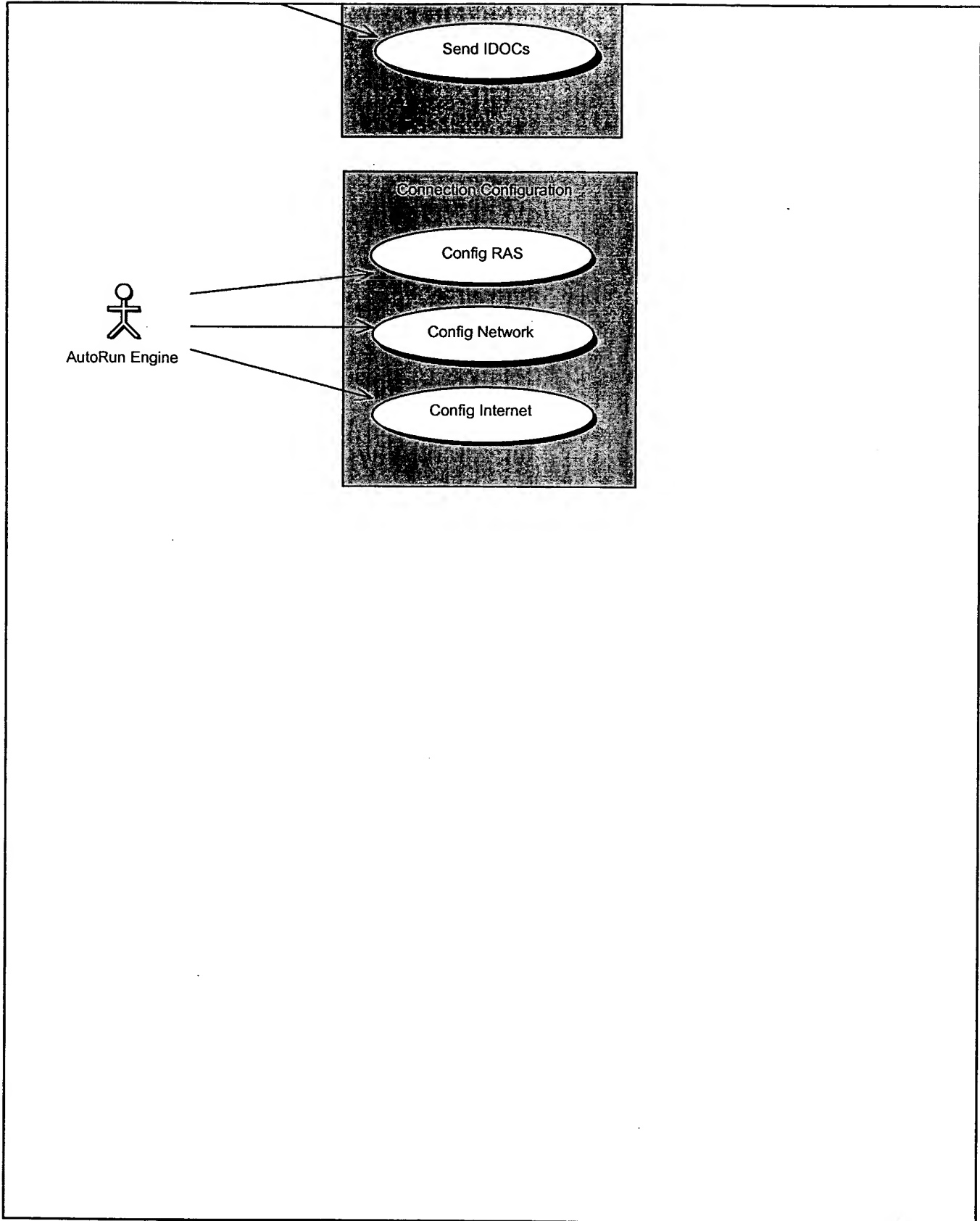
1.1, (7)CE Framework

2.1, (7)CE Framework

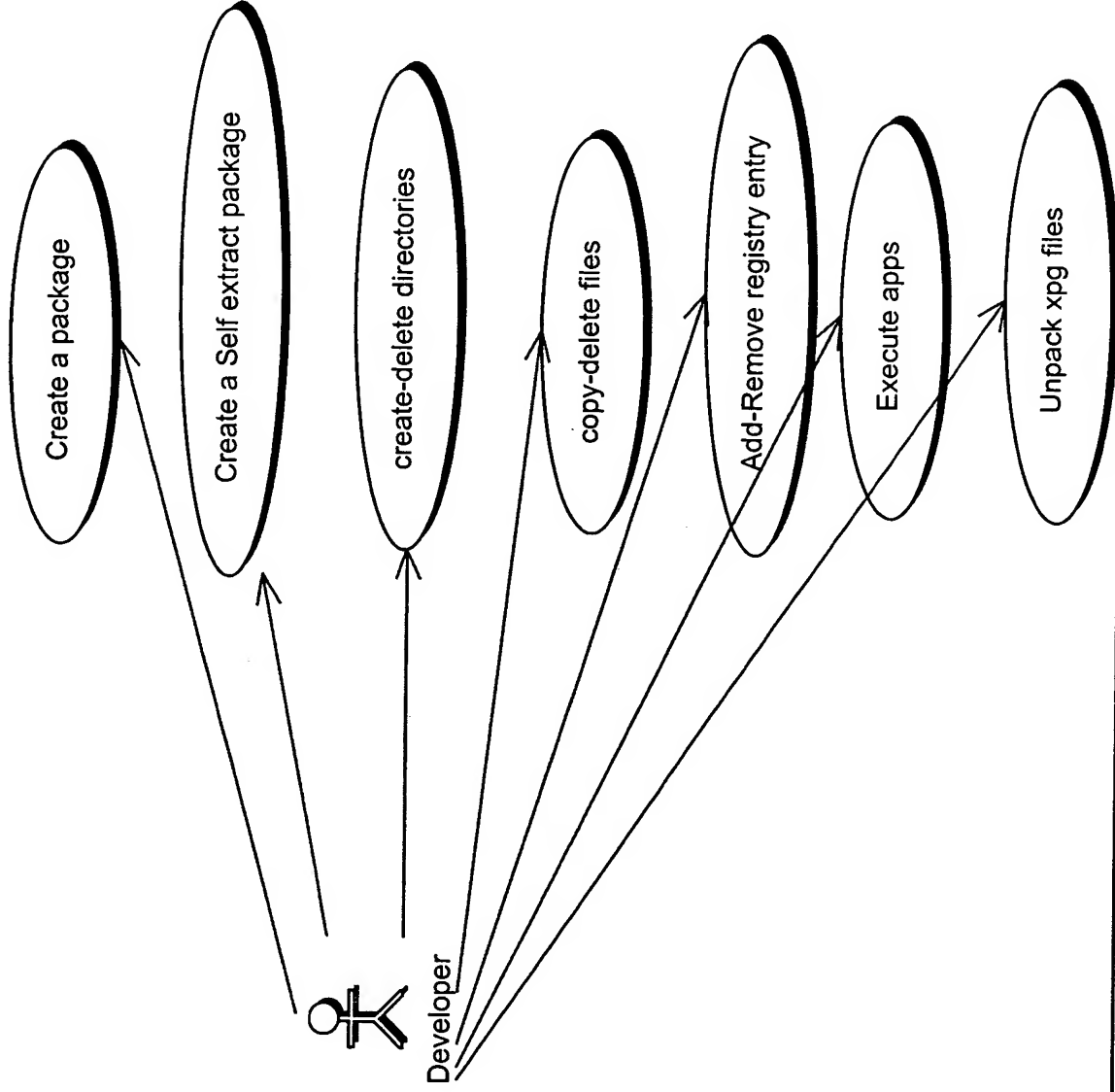


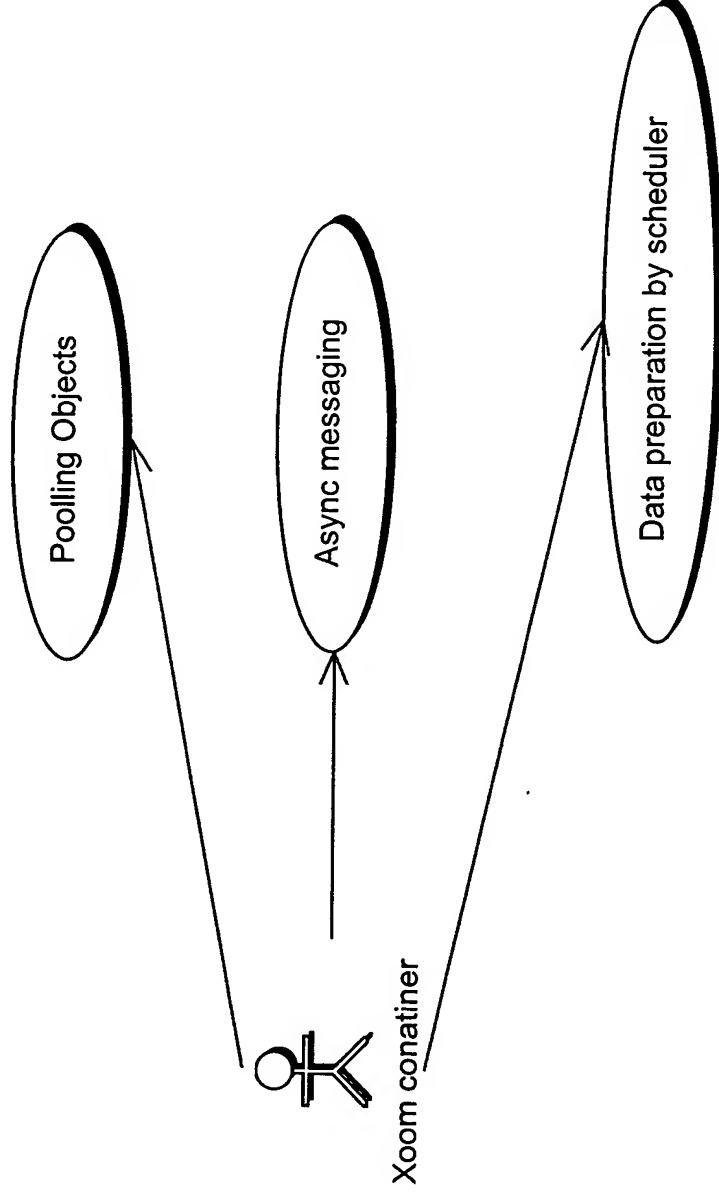
2.1, (7)CE Framework

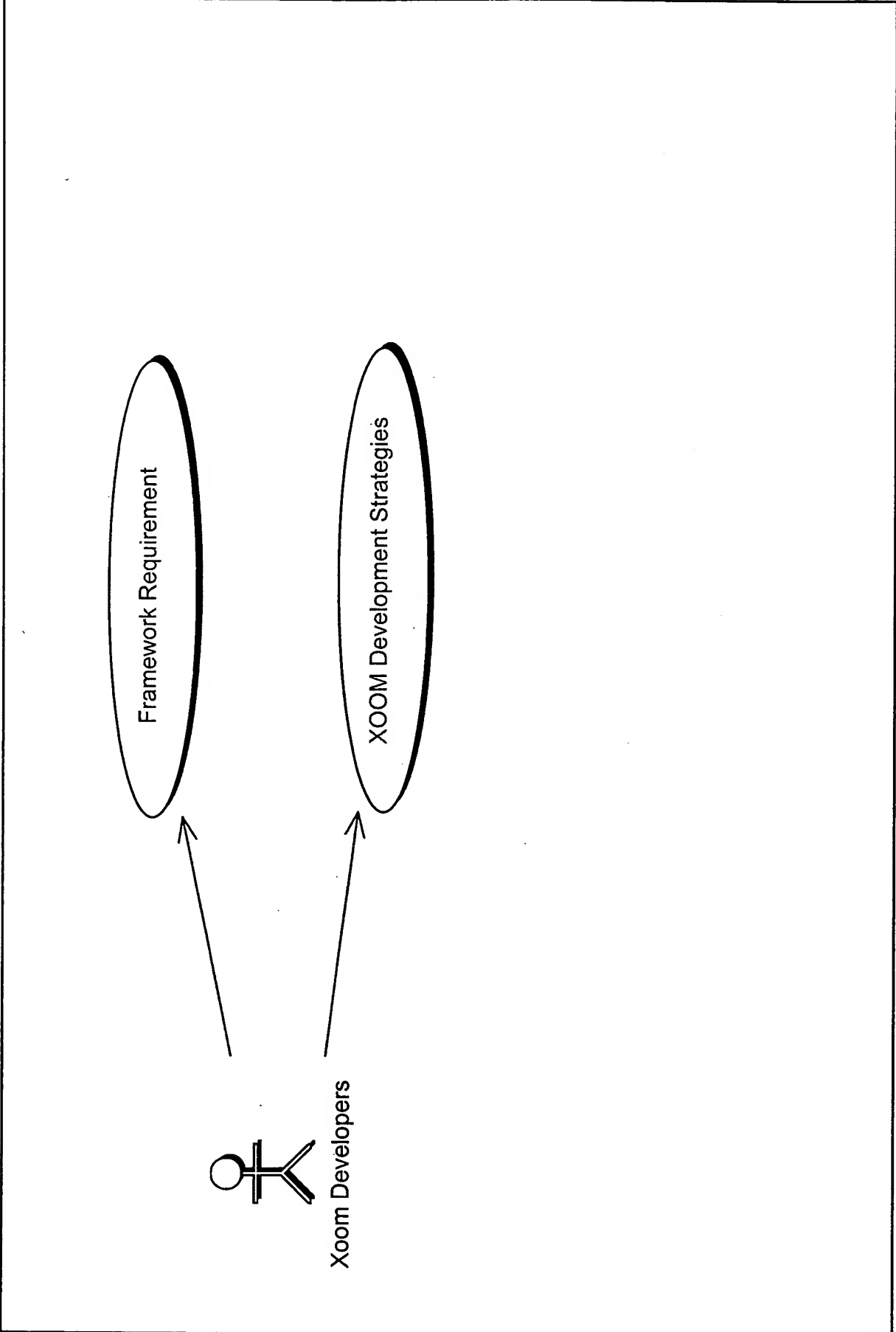
3.1, (7)CE Framework



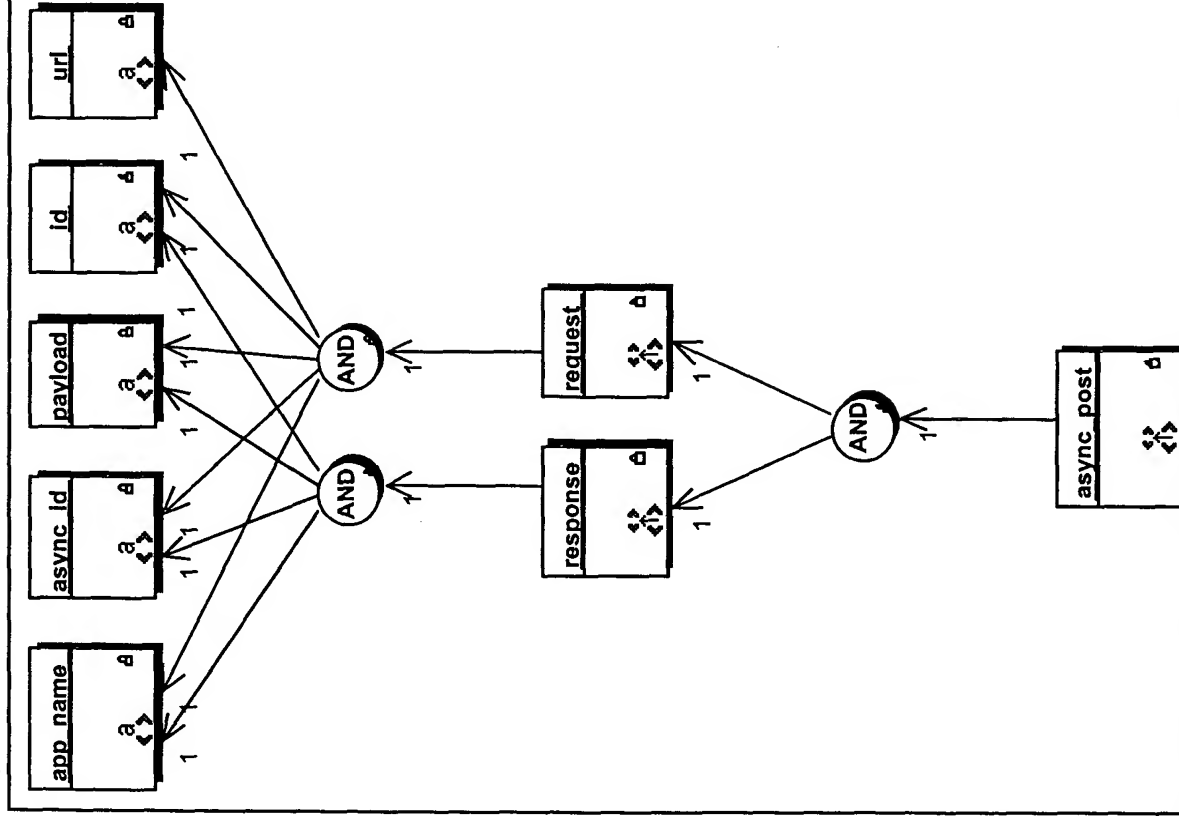
3.1, (7)CE Framework



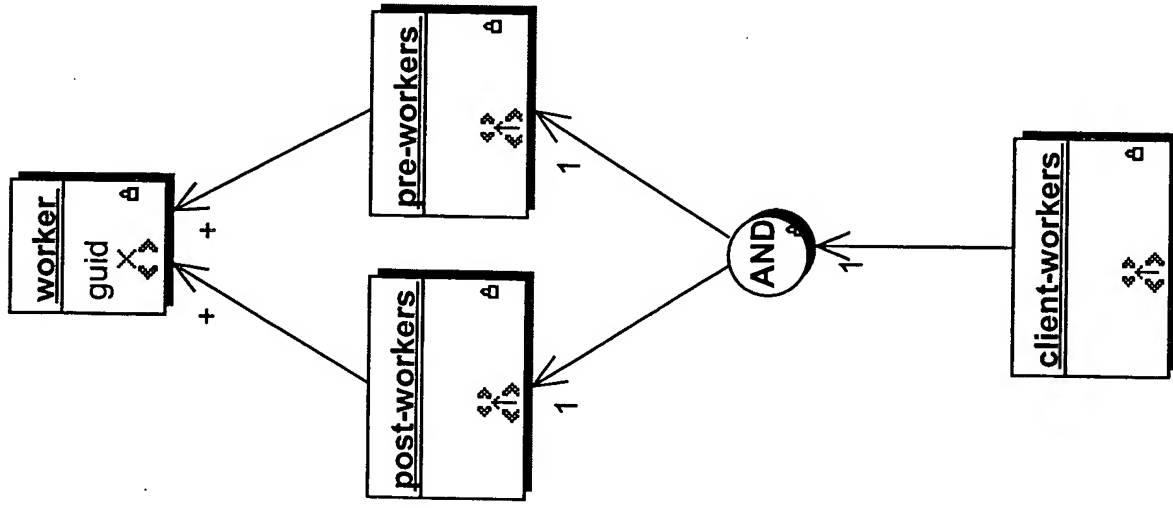




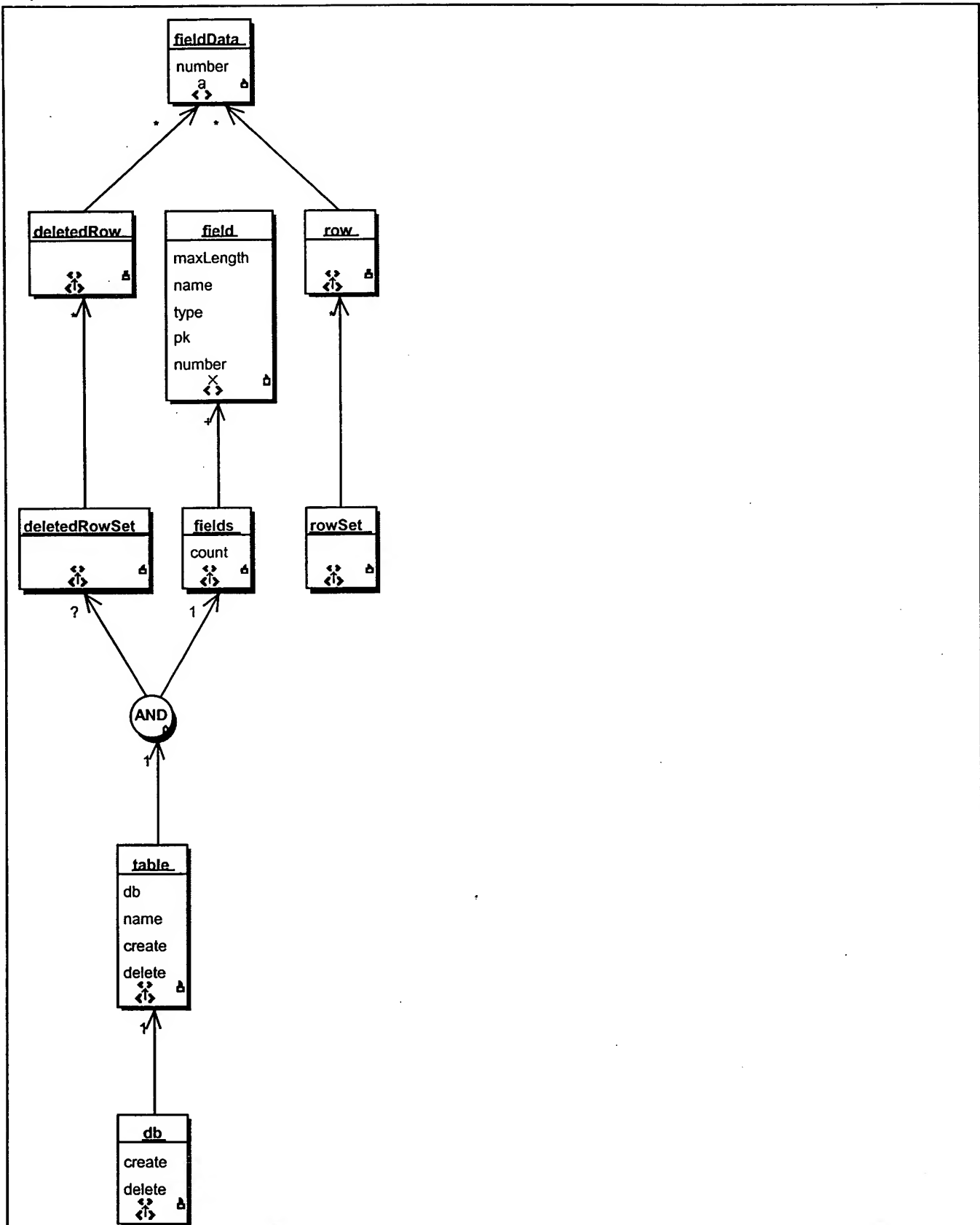
1.1, ASE XML DTD



1.1, ASE XML DTD

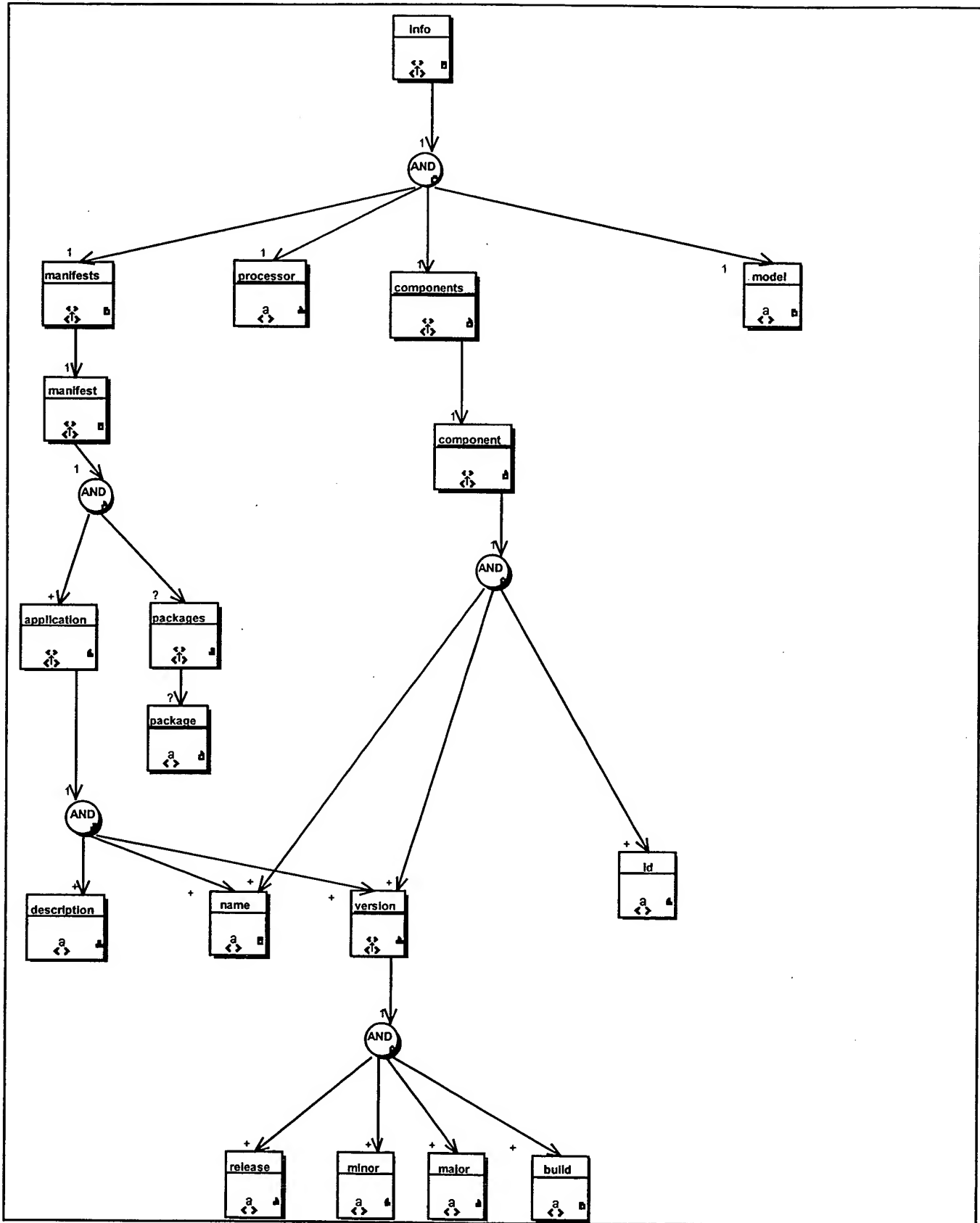


1.1, dataset XML DTD

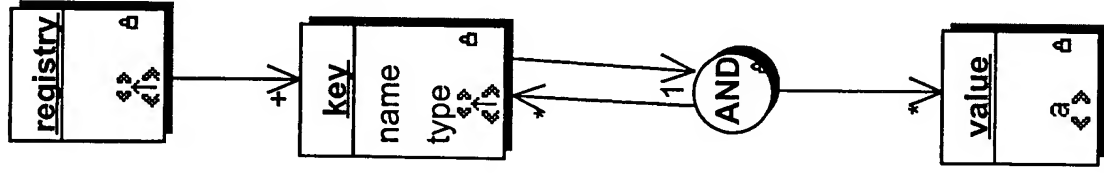


1.1, dataset XML DTD

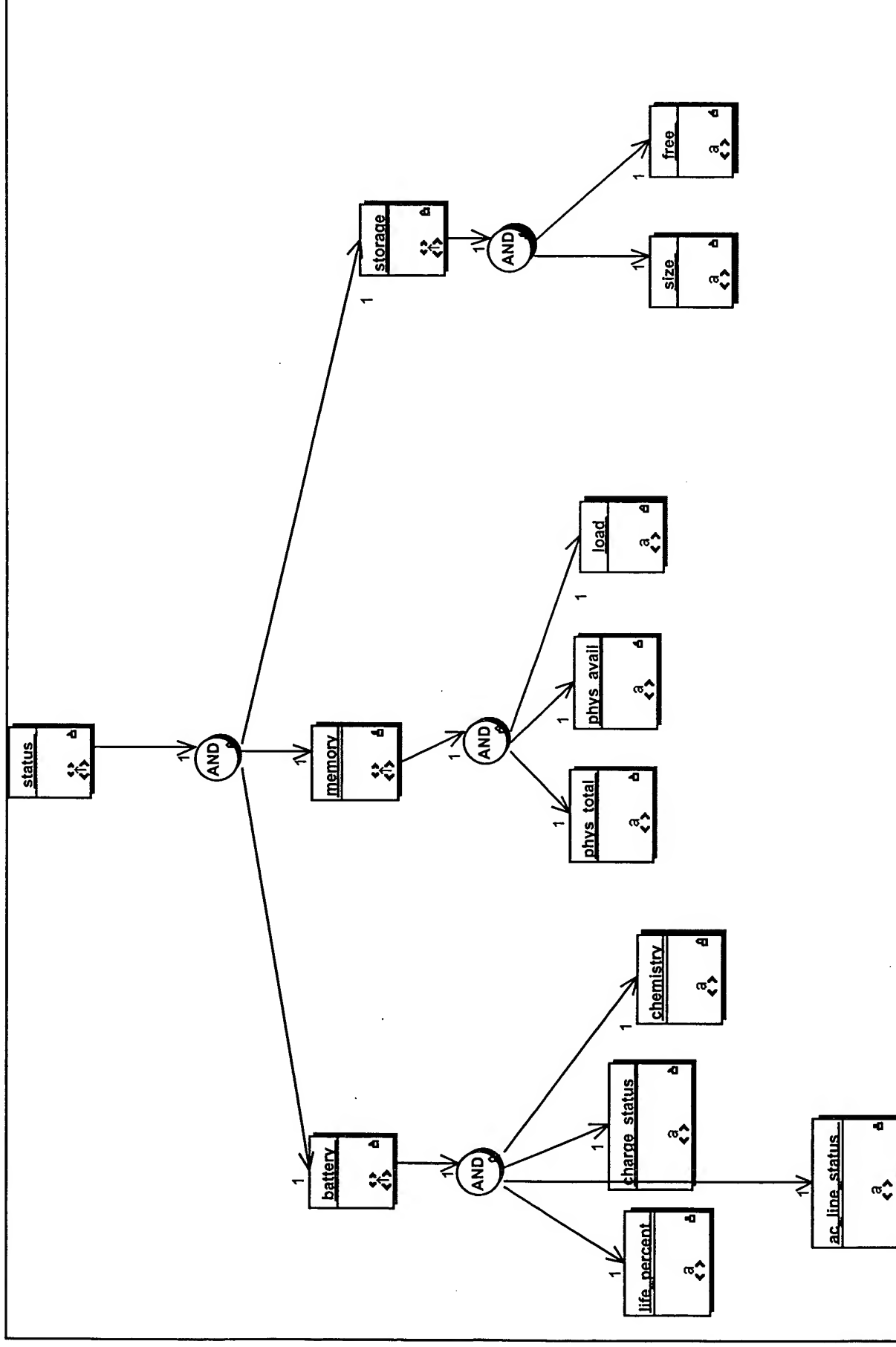
1.1, device.info XML DTD



1.1, device.info XML DTD

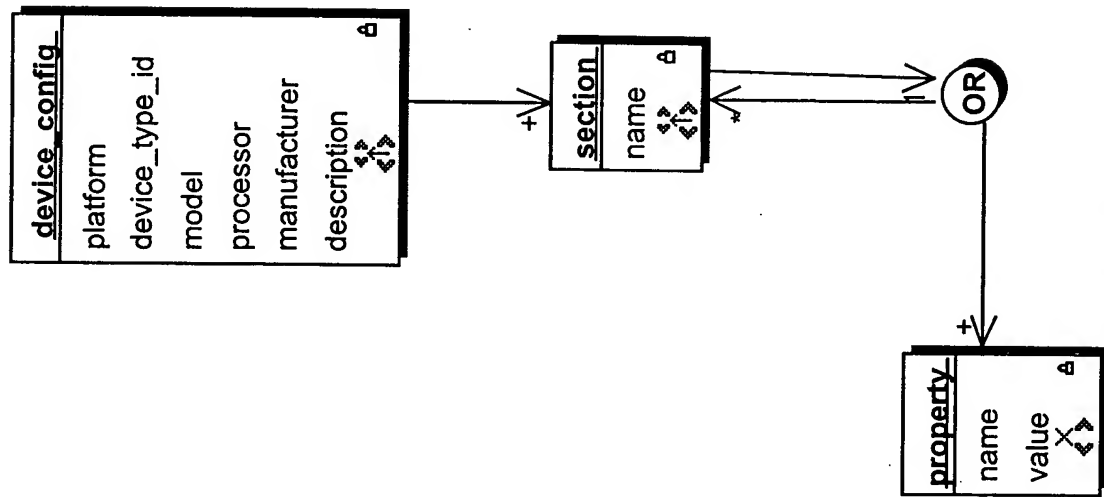


1.1.1, device.status XML DTD



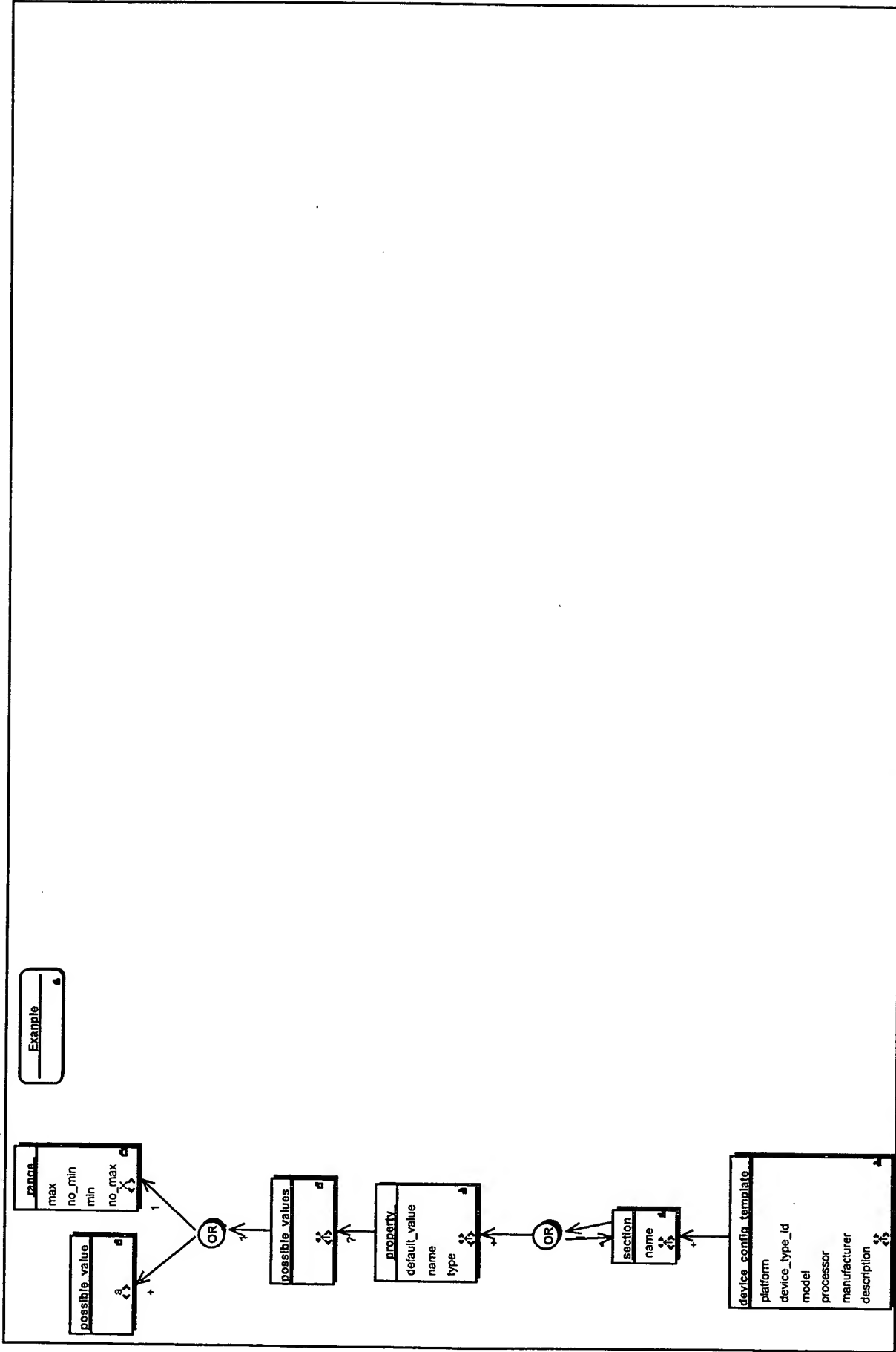
1.1.1, device.status XML DTD

1.1, device_config XML DTD

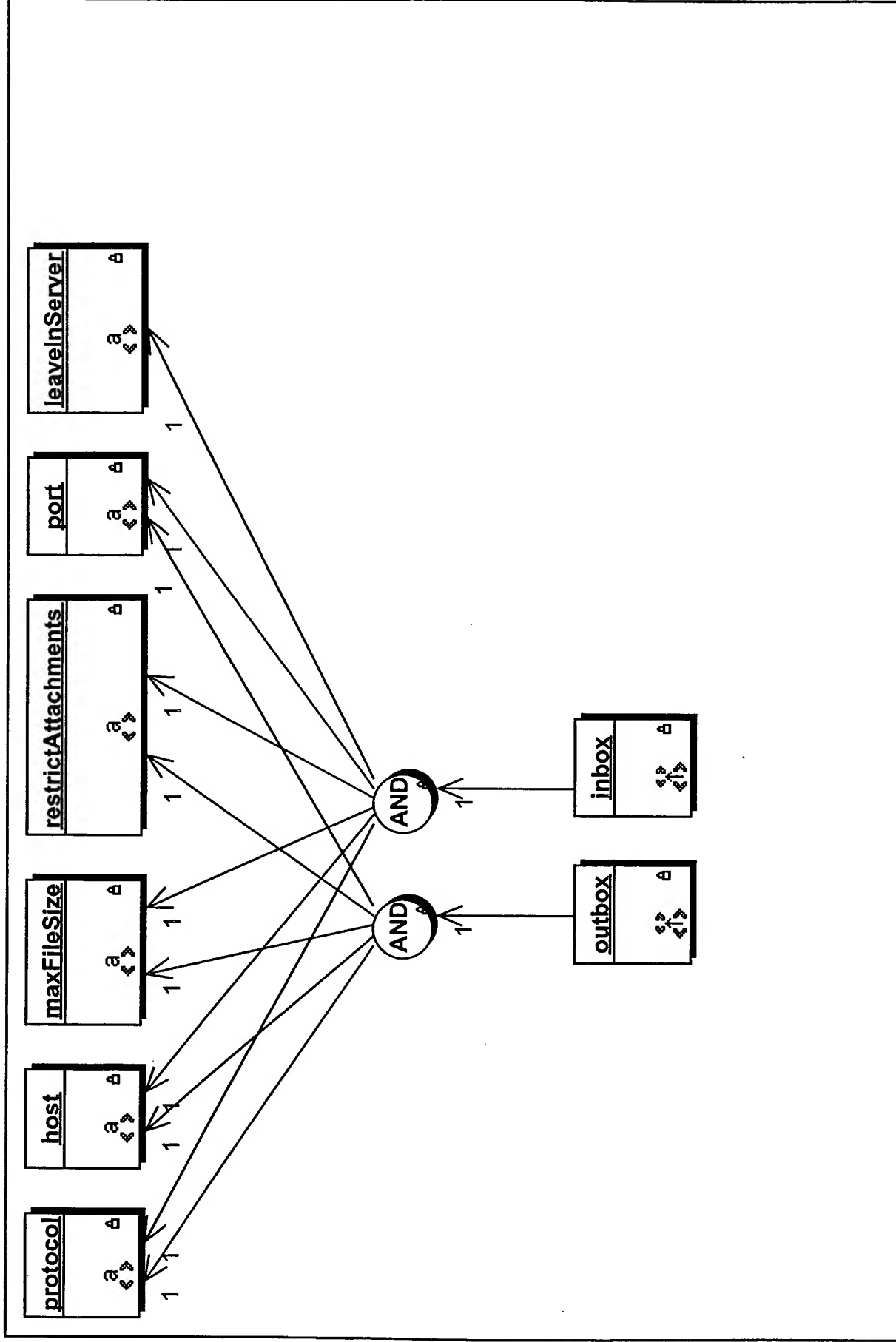


1.1, device_config XML DTD

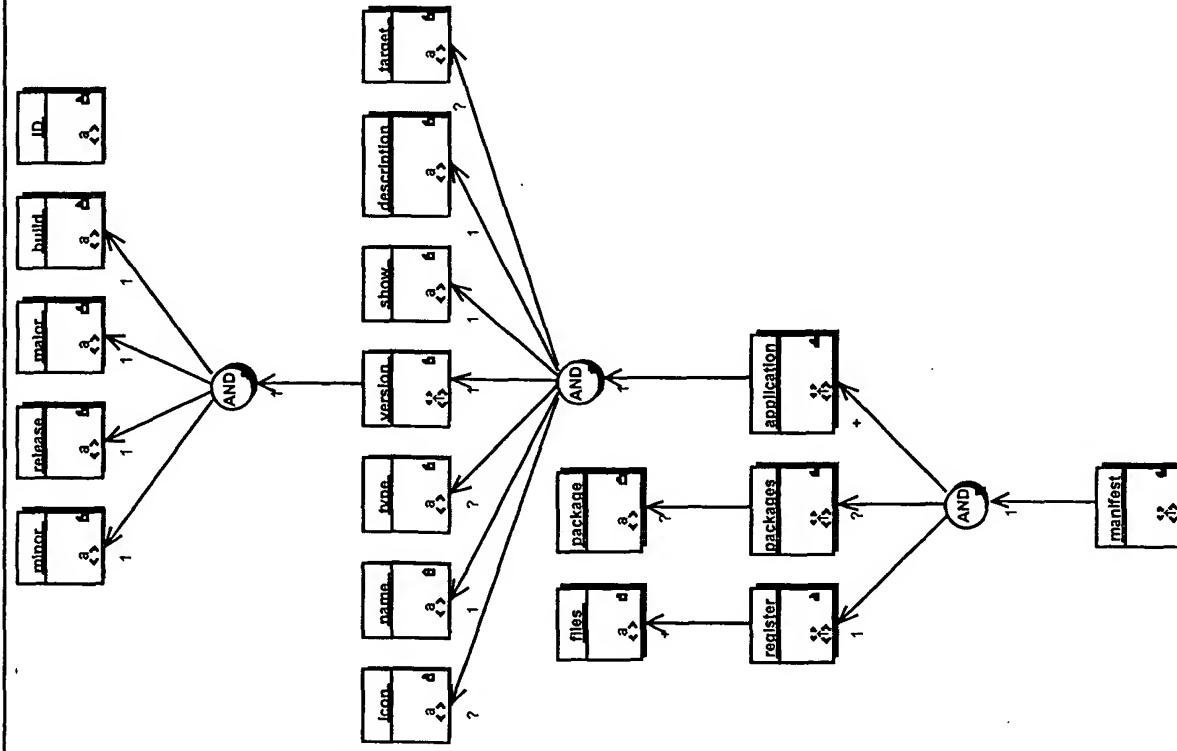
1.1.1, device_config_template XML DTD



1.1, device_config_template XML DTD

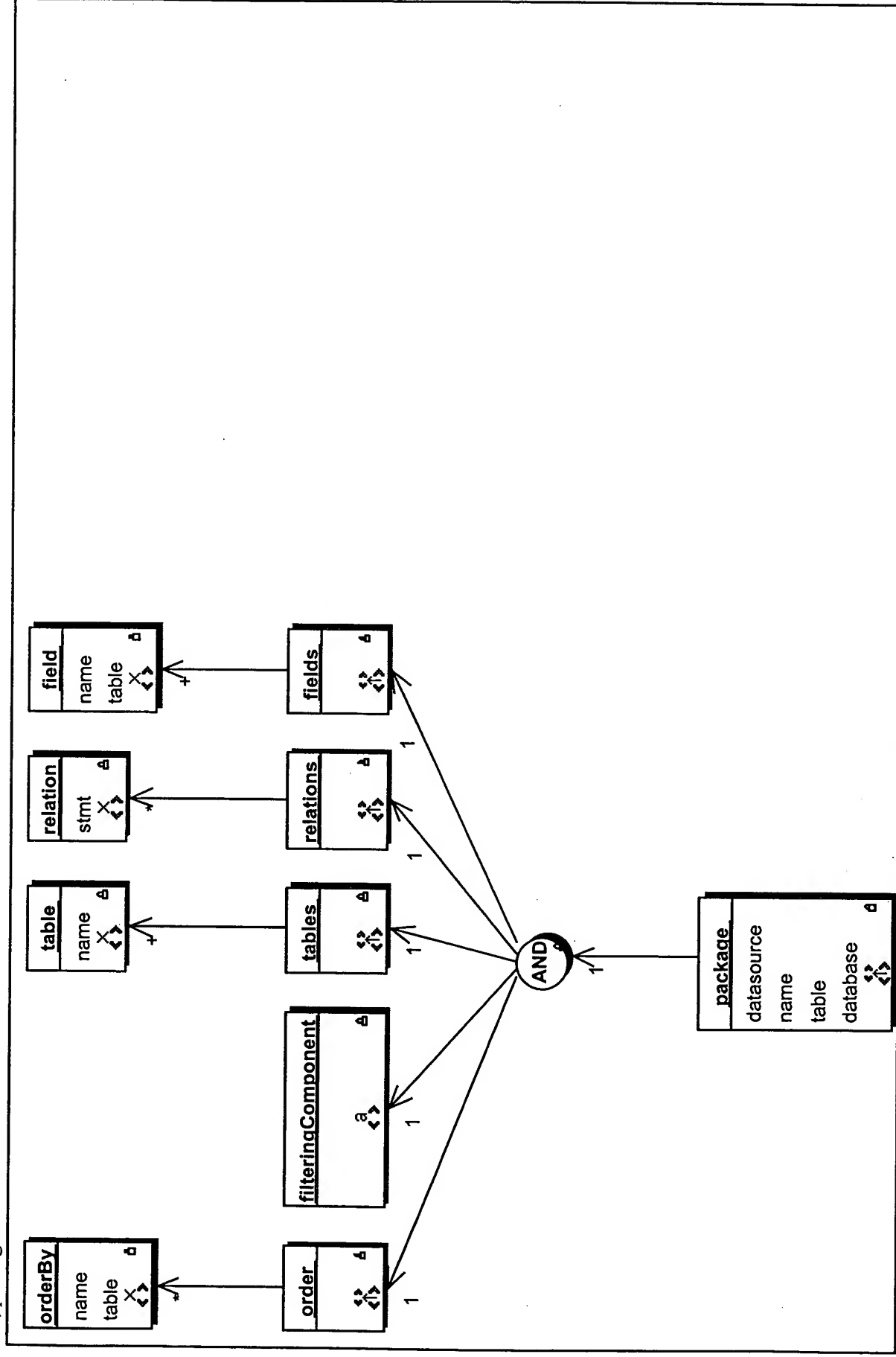


1.1, manifest XML DTD

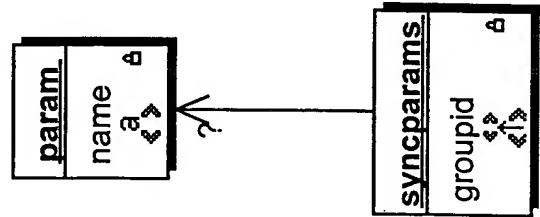


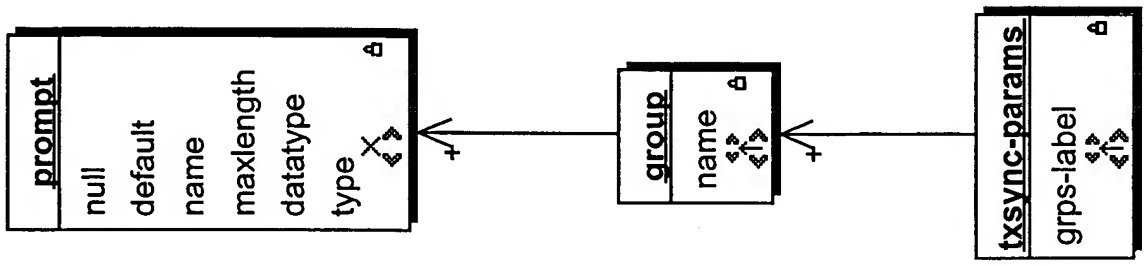
1.1, manifest XML DTD

1.1, package XML DTD

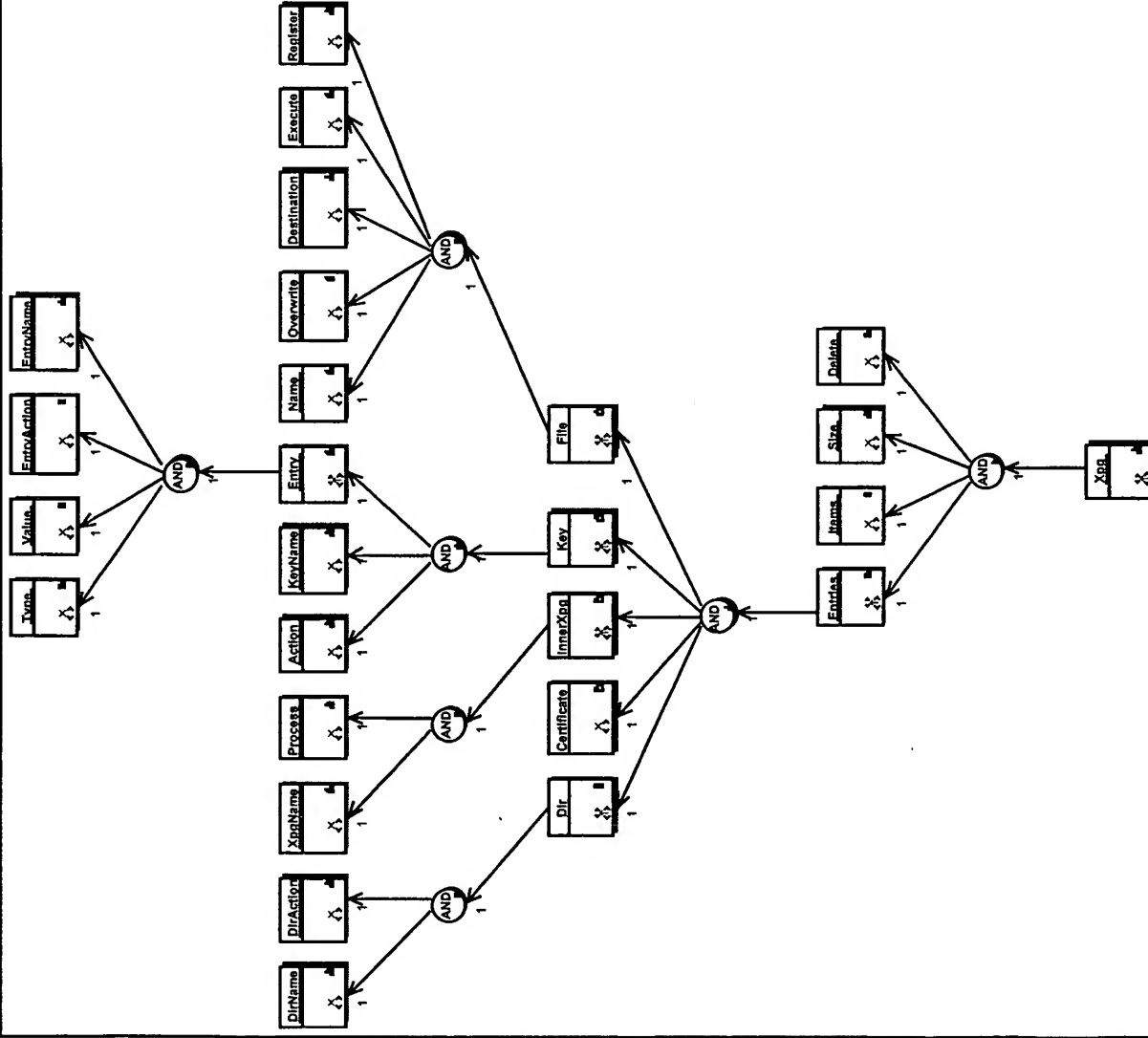


1.1, package XML DTD





1.1, xpg XML DTD



1.1, xpg XML DTD

413
414 ✓

The **Serial** control allows an application to send and receive data via the serial port of a client device. The **Serial** control provides properties to configure the communication settings of the serial port of the device and methods to send and receive data.

The Program ID for this control is **abstdio.Serial**.

| Properties | Methods | Events |
|---------------------------|-------------------------|-----------------------|
| Serial::BaudRate..... | Serial::CancelRead..... | Serial::OnDataAvailab |
| Serial::DataAvailable.... | Serial::GetData..... | le |
| Serial::DataBits | Serial::Read..... | |
| Serial::Enabled | Serial::Write..... | |
| Serial::FlowControl | | |
| Serial::InputBufferSiz | | |
| e | | |
| Serial::IsReading | | |
| Serial::Parity | | |
| Serial::Port | | |
| Serial::ReadMode | | |
| Serial::StopBits..... | | |

SECTION 250

Serial::BaudRate

The **BaudRate** property is a read/write property that determines the transmission rate of a serial connection. The **stdioBAUDRATE** enumeration lists the available baud rates. To specify a rate set the value of the **BaudRate** property to the corresponding value from the **stdioBAUDRATE** enumeration. The **BaudRate** property should be set before the serial port is opened using the **Enabled** property .

Visual Basic Syntax

```
objSerial.BaudRate = intValue
```

Error Values

ABERR_INVALIDBAUDRATE

Invalid baud rate value.

See Also

Serial::CancelRead
 Serial::Config
 Serial::DataAvailable....
 Serial::DataBits
 Serial::Enabled
 Serial::FlowControl
 Serial::GetData
 Serial::InputBufferSiz
 e
 Serial::IsReading
 Serial::OnDataAvailab
 le
 Serial::Parity
 Serial::Port
 Serial::Read
 Serial::ReadMode
 Serial::StopBits
 Serial::Write

SECTION 251

Serial::CancelRead

The **CancelRead** method stops asynchronous reading on the serial port. When the **Serial** object is set to read asynchronously, the **OnDataAvailable** (Serial) event is fired when information is received by the port. A call to the **CancelRead** method causes the object to abort reading from the port and correspondingly to stop firing the **OnDataAvailable** event.

Visual Basic Syntax

objSerial.CancelRead

See Also

Serial::BaudRate.....**Serial::Config****Serial::DataAvailable**....**Serial::DataBits****Serial::Enabled****Serial::FlowControl****Serial::GetData****Serial::InputBufferSiz****e****Serial::IsReading****Serial::OnDataAvailab****le****Serial::Parity****Serial::Port****Serial::Read****Serial::ReadMode****Serial::StopBits****Serial::Write**

SECTION 252

Serial::Config

The **Config** property is a read/write property that returns or sets the configuration scheme for the serial object. A configuration scheme consists of the values assigned to the BaudRate , DataBits , FlowControl , InputBufferSize , Parity , Port and StopBits properties. This property is not intended for application use. To set the values supplied in a configuration scheme, an application should use the corresponding properties of the Serial object.

Visual Basic Syntax

objSerial.Config = vValue

Error Values

| | |
|---------------------|--|
| ABERR_GETCONFIG | An error occurred while creating the configuration object. |
| ABERR_INVALIDCONFIG | Invalid configuration object. |
| ABERR_PUTCONFIG | An error occurred while setting the configuration object. |

See Also

Serial::BaudRate.....
 Serial::CancelRead
 Serial::DataAvailable....
 Serial::DataBits
 Serial::Enabled
 Serial::FlowControl
 Serial::GetData
 Serial::InputBufferSize
 Serial::IsReading
 Serial::OnDataAvailable
 Serial::Parity.....
 Serial::Port
 Serial::Read.....
 Serial::ReadMode
 Serial::StopBits.....
 Serial::Write.....

SECTION 253

Serial::DataAvailable

The **DataAvailable** property returns a boolean value indicating whether information has been received on the serial port or not. If the value is **true**, data has been received by the Serial port. If the value is **false**, no data has been received. If data has been received by the Serial object, the data can be accessed by calling the **GetData (Serial)** method.

Visual Basic Syntax

```
boolValue = objSerial.DataAvailable
```

See Also

Serial::BaudRate.....

Serial::CancelRead

Serial::Config

Serial::DataBits

Serial::Enabled

Serial::FlowControl

Serial::GetData

Serial::InputBufferSiz

e

Serial::IsReading

Serial::OnDataAvailab

le

Serial::Parity

Serial::Port

Serial::Read

Serial::ReadMode

Serial::StopBits.....

Serial::Write

SECTION 254

Serial::DataBits

The **DataBits** property is a read/write property that determines the number of bits that make up a character during serial communications. The `stdioDATABITS` enumeration lists the possible values for the **DataBits** property. To specify a data bits value, set the **DataBits** property to the corresponding value from the `stdioDATABITS` enumeration. The **DataBits** property should be set before the serial port is opened using the **Enabled** property.

Visual Basic Syntax

`objSerial.DataBits = intValue`

See Also

`Serial::BaudRate`.....
`Serial::CancelRead`
`Serial::Config`
`Serial::DataAvailable`....
`Serial::Enabled`
`Serial::FlowControl`
`Serial::GetData`
`Serial::InputBufferSiz`
`e`
`Serial::IsReading`
`Serial::OnDataAvailab`
`le`
`Serial::Parity`.....
`Serial::Port`
`Serial::Read`
`Serial::ReadMode`
`Serial::StopBits`.....
`Serial::Write`

SECTION 255

Serial::Enabled

The **Enabled** property can be used to open and close the serial communications port. It can also be used to query the status of the Serial object. The communication settings for the serial port are applied when the port is opened therefore it is necessary to set these settings, such as BaudRate , DataBits and FlowControl , before enabling the serial port.

To open the serial port, set the value of the Enabled property to **true**. If the Serial object is unable to open the port it will raise an error. To close the serial port, set the value of the Enabled property to **false**.

Visual Basic Syntax

objSerial.Enabled = boolValue

Error Values

| | |
|--------------------------|----------------------------------|
| ABERR_SERIAL_CREATE | Unable to create serial port. |
| ABERR_SERIAL_CONFIGURE | Unable to configure serial port. |
| ABERR_INVALIDFLOWCONTROL | Invalid flow control value. |
| ABERR_INVALIDBAUDRATE | Invalid baud rate value. |
| ABERR_INVALIDDATABITS | Invalid data bits value. |
| ABERR_INVALIDPARITY | Invalid parity value. |
| ABERR_INVALIDSTOPBITS | Invalid stop bits value. |

See Also

Serial::BaudRate.....
Serial::CancelRead
Serial::Config
Serial::DataAvailable....
Serial::DataBits
Serial::FlowControl
Serial::GetData
Serial::InputBufferSiz
e
Serial::IsReading
Serial::OnDataAvailab
le
Serial::Parity.....
Serial::Port
Serial::Read
Serial::ReadMode
Serial::StopBits.....
Serial::Write

SECTION 256

Serial::FlowControl

The **FlowControl** property is a read/write property that determines the protocol used to manage communications on the serial port. The **stdioFLOWCONTROL** enumeration lists the values corresponding to the supported flow control protocols. To specify a protocol, set the **FlowControl** property to the corresponding value from the **stdioFLOWCONTROL** enumeration. The **FlowControl** property should be set before the serial port is opened using the **Enabled** property .

Visual Basic Syntax

```
objSerial.FlowControl = intValue
```

See Also

- Serial::BaudRate.....
- Serial::CancelRead
- Serial::Config
- Serial::DataAvailable....
- Serial::DataBits
- Serial::Enabled
- Serial::GetData
- Serial::InputBufferSize
- Serial::IsReading
- Serial::OnDataAvailable
- Serial::Parity
- Serial::Port
- Serial::Read
- Serial::ReadMode
- Serial::StopBits.....
- Serial::Write

SECTION 257

Serial::GetData

The **GetData** method returns the data held in the Serial object's input buffer. Data is inserted in to the Serial object's input buffer as information is received on the serial communications port. This method returns a string value equivalent to the data currently held in the buffer. After this method is called, the data is removed from the Serial object's input buffer.

When the Read (Serial) method is called, data is appended to the Serial object's buffer as it is received. The GetData method can be called after a call to the Read method to retrieve the data.

Visual Basic Syntax

```
strValue = objSerial.GetData
```

See Also

- Serial::BaudRate.....
- Serial::CancelRead
- Serial::Config
- Serial::DataAvailable....
- Serial::DataBits
- Serial::Enabled
- Serial::FlowControl
- Serial::InputBufferSiz
- e
- Serial::IsReading
- Serial::OnDataAvailab
- le
- Serial::Parity
- Serial::Port
- Serial::Read
- Serial::ReadMode
- Serial::StopBits.....
- Serial::Write

SECTION 258

Serial::InputBufferSize

The **InputBufferSize** property is a read/write property that sets or retrieves the size of the serial port's input buffer. This property determines the maximum number of characters that can be read from the serial port during a single read operation. This property must be set before the Serial object is enabled.

During synchronous reading, the Serial object performs a single read operation from the communications port when the Read (Serial) method is called. The maximum number of characters that can be received with this single read operation is equal to the InputBufferSize. When reading asynchronously, the Serial object may perform multiple read operations. As each read operation is completed, data is moved from the serial communication port's input buffer and appended to a second buffer controlled by the Serial object. This second buffer is not limited by the InputBufferSize setting.

The default value for this property is 512.

Visual Basic Syntax

objSerial.InputbufferSize = intValue

Error Values

ABERR_INVALIDBUFFERSIZE

Invalid buffer size.

See Also

Serial::BaudRate.....
 Serial::CancelRead
 Serial::Config.....
 Serial::DataAvailable....
 Serial::DataBits
 Serial::Enabled
 Serial::FlowControl
 Serial::GetData
 Serial::IsReading
 Serial::OnDataAvailab
 le.....
 Serial::Parity.....
 Serial::Port
 Serial::Read.....
 Serial::ReadMode.....
 Serial::StopBits.....
 Serial::Write.....

SECTION 259

Serial::IsReading

The **IsReading** property returns a value indicating the read status of the Serial object. If the value is **true** the Serial object is currently reading data from the port. If the value is false, the Serial object is not reading data from the communications port. This property is only applicable when reading asynchronously.

Visual Basic Syntax

```
boolValue = objSerial.IsReading
```

See Also

- Serial::BaudRate.....
- Serial::CancelRead
- Serial::Config
- Serial::DataAvailable....
- Serial::DataBits
- Serial::Enabled
- Serial::FlowControl
- Serial::GetData
- Serial::InputBufferSiz
e
- Serial::OnDataAvailab
le
- Serial::Parity
- Serial::Port
- Serial::Read
- Serial::ReadMode
- Serial::StopBits
- Serial::Write

SECTION 260

Serial::OnDataAvailable

The **OnDataAvailable** event is fired when the Serial object successfully reads data from the serial port. This event is only fired when reading is performed asynchronously. The `GetData` method can be used to retrieve the data read from the port as a string.

Visual Basic Syntax

```
Set objSerial = CreateObjectWithEvents("abstdio.Serial", "Serial_")
```

```
...
```

```
Sub Serial_OnDataAvailable()
```

```
...
```

```
End Sub
```

See Also

Serial::BaudRate.....

Serial::CancelRead

Serial::Config

Serial::DataAvailable....

Serial::DataBits

Serial::Enabled

Serial::FlowControl

Serial::GetData

Serial::InputBufferSiz

e

Serial::IsReading

Serial::Parity

Serial::Port

Serial::Read

Serial::ReadMode

Serial::StopBits

Serial::Write

SECTION 261

Serial::Parity

The **Parity** property is a read/write property that determines the algorithm used to set the error checking bit of a character during serial communications. The **stdioPARITY** enumeration lists the possible values corresponding to each parity setting. To set the parity, assign the **Parity** property to the corresponding value from the **stdioPARITY** enumeration. The **Parity** property should be set before the serial port is opened using the **Enabled** property.

Visual Basic Syntax

`objSerial.Parity = intValue`

See Also

`Serial::BaudRate`

`Serial::CancelRead`

`Serial::Config`

`Serial::DataAvailable`....

`Serial::DataBits`

`Serial::Enabled`

`Serial::FlowControl`

`Serial::GetData`

`Serial::InputBufferSiz`

`e`

`Serial::IsReading`

`Serial::OnDataAvailab`

`le`

`Serial::Port`

`Serial::Read`

`Serial::ReadMode`

`Serial::StopBits`.....

`Serial::Write`

SECTION 262

Serial::Port

The **Port** property is a read/write property that determines the communications hardware used by the serial object. The `stdioPORT` enumeration lists the possible setting for the Port property. To specify a port set the value of the Port property to the corresponding value from the `stdioPORT` enumeration. The Port property should be set before the serial port is opened using the Enabled property. The default value of this property is "port one" (COM1).

Visual Basic Syntax

```
objSerial.Port = intValue
```

See Also

`Serial::BaudRate`.....

`Serial::CancelRead`

`Serial::Config`

`Serial::DataAvailable`....

`Serial::DataBits`

`Serial::Enabled`

`Serial::FlowControl`

`Serial::GetData`

`Serial::InputBufferSiz`

`e`

`Serial::IsReading`

`Serial::OnDataAvailab`

`le`

`Serial::Parity`.....

`Serial::Read`

`Serial::ReadMode`

`Serial::StopBits`.....

`Serial::Write`

SECTION 263

Serial::Read

The **Read** method is used to start receiving information from the serial communications port. This method takes two parameters. The first parameter, **ReadMode**, is a value indicating the type of read operation to be executed. The second parameter, **Seconds**, specifies the length of time that a synchronous read operation will be executed.

The possible values of the **ReadMode** parameter are defined in the **stdio_READMODE** enumeration. The two possible types of read operations are synchronous and asynchronous. When a synchronous read is selected, the **Serial** object performs a single read operation. When this read operation is completed, the **Read** method returns with the data read from the port assigned to the output value of the method.

When an asynchronous read is selected, the **Serial** object will begin performing read operations indefinitely and the **Read** method will return immediately. The **OnDataAvailable (Serial)** event will be fired as each successful read operation is completed. The **CancelRead** method can be used to stop asynchronous reading from the serial port. When an asynchronous read is specified, the read method will not return any values read from the serial port.

Visual Basic Syntax

```
strValue=objSerial.Read( stdioReadMode, intSeconds )
```

See Also

Serial::BaudRate.....
Serial::CancelRead
Serial::Config
Serial::DataAvailable....
Serial::DataBits
Serial::Enabled
Serial::FlowControl
Serial::GetData
Serial::InputBufferSiz
e
Serial::IsReading
Serial::OnDataAvailab
le
Serial::Parity.....
Serial::Port
Serial::ReadMode
Serial::StopBits.....
Serial::Write

SECTION 264

Serial::ReadMode

The **ReadMode** property returns the current read setting of the Serial object. The possible values returned by the ReadMode property are specified in the `stdio_READMODE` enumeration. The possible read settings of the Serial object are *synchronous* and *asynchronous*. See the Read (Serial) method for more information on the types of read settings.

Visual Basic Syntax

```
intValue = objSerial.ReadMode
```

See Also

Serial::BaudRate.....
 Serial::CancelRead
 Serial::Config
 Serial::DataAvailable....
 Serial::DataBits
 Serial::Enabled
 Serial::FlowControl
 Serial::GetData
 Serial::InputBufferSiz
 e
 Serial::IsReading
 Serial::OnDataAvailab
 le
 Serial::Parity
 Serial::Port
 Serial::Read
 Serial::StopBits.....
 Serial::Write

SECTION 265

Serial::StopBits

The **StopBits** property is a read/write property that determines how the end of a character is signalled during serial communications. The possible values of the StopBits property are specified in the `stdioSTOPBITS` enumeration. To set the number of stop bits, assign the StopBits property to the corresponding value from the `stdioSTOPBITS` enumeration. The StopBits property should be set before the serial port is opened using the Enabled property .

Visual Basic Syntax

```
objSerial.StopBits = intValue
```

See Also

Serial::BaudRate.....
 Serial::CancelRead
 Serial::Config
 Serial::DataAvailable....
 Serial::DataBits
 Serial::Enabled
 Serial::FlowControl
 Serial::GetData
 Serial::InputBufferSiz
 e
 Serial::IsReading
 Serial::OnDataAvailab
 le
 Serial::Parity
 Serial::Port
 Serial::Read
 Serial::ReadMode
 Serial::Write

SECTION 266

Serial::Write

The **Write** method is used to send data over the serial communications port. The Write method has one string parameter that holds the data to be sent using the Serial object. A serial communications session must be opened using the Enabled property before data can be successfully written using the port.

If the Serial object is not able to write the data using the port, an error is thrown.

Visual Basic Syntax

```
objSerial.Write( strData )
```

Error Values

ABERR_SERIALWRITE

An error occurred while writing to the Seri

See Also

Serial::BaudRate.....
 Serial::CancelRead
 Serial::Config
 Serial::DataAvailable....
 Serial::DataBits
 Serial::Enabled
 Serial::FlowControl
 Serial::GetData
 Serial::InputBufferSiz
 e
 Serial::IsReading
 Serial::OnDataAvailab
 le
 Serial::Parity.....
 Serial::Port
 Serial::Read
 Serial::ReadMode
 Serial::StopBits.....

abSys (System Core)

The **abSys** library provides access to the core functionality of the client device framework. This library provides methods for discovery of the installed applications on the client device and triggering of the TxSync process. The abSys library is a C++ dynamic link library and can be accessed from eMbedded Visual Basic applications using the `Declare` statement. C/C++ applications can access the library in Windows CE by using the `LoadLibrary` statement.

Visual Basic Syntax

```
Declare Function FindFirstApp Lib"absys" Alias "AbFindFirstApp" () As  
Long
```

```
Declare Function FindNextApp Lib"absys" Alias "AbFindNextApp" () As  
Long
```

SECTION 267

ContinueSync

The **AbContinueSync** function is used to resume the TxSync process when it has timed out. The function takes the address of a TxSyncCallbackFunc function as a parameter. This callback function will be called as the status of the synchronization changes. This function should only be called after a previous call to AbExecuteSync or if it is known that a synchronization is active.

The AbContinueSync function returns a status code indicating the state of the synchronization when the process completes.

Visual Basic Syntax

Declare Function AbContinueSync Lib"absys" (ByVal lpStatusCallback As Long) As Long

C++ Syntax

int AbContinueSync(TxSyncCallbackFunc lpStatusCallback)

See Also

ContinueSyncUI

ExecuteSync

ExecuteSyncUI

AbFindFirstApp

AbFindNextApp

AbGetAppProperty

AbDestroyUI

AbGetAppPropertyW ...

AbGetAppCount

AbGetSyncInfo

AbGetSyncLog

TxSyncCallbackFunc....

SECTION 268

ContinueSyncUI

The **AbContinueSyncUI** function is used to resume the TxSync process when it has timed out. The AbContinueSyncUI function provides essentially the same functionality provided by the AbContinueSync function. The difference is that AbContinueSyncUI function displays a status window indicating the state of the TxSync process as it is executed. The hwndParent parameter can be used to specify a window from the calling process that will act as the parent to any status windows created during the function call.

The bDestroyUI parameter is a flag that indicates whether any windows created during the function call should be automatically destroyed when the function returns. A value of **true** indicates that any windows created will be destroyed when the function returns. A value of **false** indicates that the calling process will take responsibility for destroying any windows that are created. Any created windows can be destroyed using the AbDestroyUI function.

Visual Basic Syntax

```
Declare Function AbContinueSyncUI Lib"absys" (ByVal hwndParent As
Long, ByVal bDestroyUI As Boolean) As Long
```

C++ Syntax

```
int AbContinueSyncUI(HWND hwndParent, bool bDestroyUI )
```

See Also

ContinueSync
 ExecuteSync
 ExecuteSyncUI
 AbFindFirstApp
 AbFindNextApp
 AbGetAppProperty
 AbDestroyUI
 AbGetAppPropertyW ...
 AbGetAppCount
 AbGetSyncInfo
 AbGetSyncLog
 TxSyncCallbackFunc....

SECTION 269

ExecuteSync

The **AbExecuteSync** function is used to launch the TxSync process. The first parameter to this function is a string containing the synchronization parameters for the current synchronization. The second parameter to this function is the address of a callback function.

The synchronization parameters are formatted as an XML document. These parameters include the group selected for the synchronization and the names and values of the parameters associated with the indicated group.

The callback function passed during a call to **AbExecuteSync** will be called when the TxSync process has new status information to report to the caller. See **TxSyncCallbackFunc** for a description of how this function is implemented.

Visual Basic Syntax

Declare Function **AbExecuteSync** Lib"absys" (ByVal strGroupParams As String, ByVal lpStatusCallback As Long) As Long

C++ Syntax

int **AbExecuteSync**(BSTR strGroupParams, TxSyncCallbackFunc lpStatusCallback)

When this function is called the complete TxSync process is executed. This function will not return until the process has completed. The value returned is a status code indicating the final status of the synchronization at the time of completion. The callback function may be called several times during execution of the TxSync process.

See Also

ContinueSync
ContinueSyncUI
ExecuteSyncUI
AbFindFirstApp
AbFindNextApp
AbGetAppProperty
AbDestroyUI
AbGetAppPropertyW ...
AbGetAppCount
AbGetSyncInfo
AbGetSyncLog
TxSyncCallbackFunc

SECTION 270

ExecuteSyncUI

The **AbExecuteSyncUI** function is used to launch the TxSync process and display a status window that is updated as the process executes. The **hwndParent** parameter may be used to pass a window handle from the calling process that will be specified as the parent window of any windows created by the **AbExecuteSyncUI** function. The second parameter is a string containing the synchronization parameters XML document. The third boolean parameter is a flag to indicate whether any status windows that are created should be automatically destroyed. The operation of this function is identical to the **AbExecuteSync** function.

Note that if this function returns a status code indicating that the synchronization is complete (i.e. aborted or succeeded), any status windows created will be destroyed regardless of the value of the **bDestroyUI** parameter.

Visual Basic Syntax

```
Declare Function AbExecuteSyncUI Lib"absys" (ByVal hwndParent As
Long, ByVal strGroupParams As String, ByVal bDestroyUI As Boolean) As
Long
```

C++ Syntax

```
int AbExecuteSyncUI(HWND hwndParent, BSTR strGroupParams, bool
bDestroyUI )
```

See Also

ContinueSync
ContinueSyncUI
ExecuteSync
AbFindFirstApp.....
AbFindNextApp
AbGetAppProperty
AbDestroyUI
AbGetAppPropertyW ...
AbGetAppCount.....
AbGetSyncInfo.....
AbGetSyncLog.....
TxSyncCallbackFunc....

SECTION 271

AbFindFirstApp

The **AbFindFirstApp** function is used to initialize a new search for applications that have been installed on the client device. This function returns an integer code that indicates the result of the new search for the application. If an application has been found, the **AbGetAppProperty** function can be used to retrieve information regarding the application that has been found.

After a first application has been found, the **AbFindNextApp** function can be called to find another application that has been installed on the client device. The **AbFindNextApp** function can be called iteratively to find each application on the client device until the function returns a code indicating that the last application has been found.

Visual Basic Syntax

```
Declare Function FindFirstApp Lib"absys" Alias "AbFindFirstApp" () As Long
```

C++ Syntax

```
int AbFindFirstApp()
```

See Also

- ContinueSync
- ContinueSyncUI
- ExecuteSync
- ExecuteSyncUI
- AbFindNextApp
- AbGetAppProperty
- AbDestroyUI
- AbGetAppPropertyW ...
- AbGetAppCount
- AbGetSyncInfo
- AbGetSyncLog
- TxSyncCallbackFunc....

SECTION 272

AbFindNextApp

The **AbFindNextApp** function is used to continue a search for applications installed on a client device. This function returns an integer code indicating the result of the search for the application. If an application has been found, the **AbGetAppProperty** function can be used to retrieve information regarding the application that has been found.

The **AbFindNextApp** function can be called repeatedly to iterate through the applications installed on the client device.

Visual Basic Syntax

Declare Function **AbFindNextApp** Lib"absys" () As Long

C++ Syntax

int **AbFindNextApp**()

See Also

ContinueSync

ContinueSyncUI

ExecuteSync

ExecuteSyncUI

AbFindFirstApp

AbGetAppProperty

AbDestroyUI

AbGetAppPropertyW ...

AbGetAppCount

AbGetSyncInfo

AbGetSyncLog

TxSyncCallbackFunc....

SECTION 273

AbGetAppProperty

The **AbGetAppProperty** function is used to retrieve information on an application installed on the client device. This function should only be called after a call to the **AbFindFirstApp** or **AbFindNextApp** functions has returned a code indicating that an application has been found.

Each property of an application can be retrieved by the property name or key. The **szKey** parameter specifies this name. The value of the property will be returned in the **lpszValue** parameter. The **lpdwSize** parameter is used to indicate the maximum number of characters that can be held in the **lpszValue** buffer passed to the function. If this buffer is insufficient, the function will return the number of required characters in this parameter.

Embedded Visual Basic developers should use the **AbGetAppPropertyW** function instead of the **AbGetAppProperty** function.

C++ Syntax

```
int AbGetAppProperty(LPCTSTR szKey As String, LPSTR lpszValue As
String, LPDWORD lpcbSize)
```

See Also

ContinueSync
ContinueSyncUI
ExecuteSync
ExecuteSyncUI
AbFindFirstApp
AbFindNextApp
AbDestroyUI
AbGetAppPropertyW ...
AbGetAppCount
AbGetSyncInfo
AbGetSyncLog
TxSyncCallbackFunc....

SECTION 274

AbDestroyUI

The **AbDestroyUI** function is used to destroy any status windows that may have been created during a call to the **AbContinueSyncUI** or **AbExecuteSyncUI** functions. When the **AbDestroyUI** function is called any windows created during previous calls to **AbContinueSyncUI** or **AbExecuteSyncUI** are closed and any associated resources are released.

Visual Basic Syntax

Declare Sub **AbDestroyUI** Lib "absys" ()

C++ Syntax

void **AbDestroyUI**()

See Also

ContinueSync

ContinueSyncUI

ExecuteSync

ExecuteSyncUI

AbFindFirstApp

AbFindNextApp

AbGetAppProperty

AbGetAppPropertyW ...

AbGetAppCount

AbGetSyncInfo

AbGetSyncLog

TxSyncCallbackFunc....

SECTION 275

AbGetAppPropertyW

The **AbGetAppPropertyW** function provides the same functionality as the **AbGetAppProperty** function and is used in the same manner. The value of the property requested is returned as an automation string in the **bstrValue** parameter for ease of use from languages that make heavy use of automation such as Visual Basic.

Visual Basic Syntax

Declare Function **AbGetAppPropertyW** Lib "absys" (ByVal **szKey** As String, **bstrValue** As String, **lpdwSize** As Long) As Long

C++ Syntax

int **AbGetAppPropertyW** (LPCTSTR **szKey** As String, BSTR* **bstrValue** As String, LPDWORD **lpcbSize**)

See Also

ContinueSync
ContinueSyncUI
ExecuteSync
ExecuteSyncUI
AbFindFirstApp
AbFindNextApp
AbGetAppProperty
AbDestroyUI
AbGetAppCount
AbGetSyncInfo
AbGetSyncLog
TxSyncCallbackFunc....

SECTION 276

AbGetAppCount

The **AbGetAppCount** function returns the total number of applications installed on the client device. When called the **AbGetAppCount** function searches the client device and returns the total number of applications installed by the system. This count of applications does not include applications that were not installed by the system.

Visual Basic Syntax

Declare Function AbGetAppCount Lib "absys" () As Long

C++ Syntax

int AbGetAppCount()

See Also

ContinueSync
ContinueSyncUI
ExecuteSync
ExecuteSyncUI
AbFindFirstApp
AbFindNextApp
AbGetAppProperty
AbDestroyUI
AbGetAppPropertyW ...
AbGetSyncInfo
AbGetSyncLog
TxSyncCallbackFunc....

SECTION 277

AbGetSyncInfo

The **AbGetSyncInfo** function retrieves the latest version of the Group Parameters document from an Atoma server. The Group Parameters information is formatted as an XML document and is returned in the pstrXML parameter.

Visual Basic Syntax

Declare Function AbGetSyncInfo Lib "absys" (pstrXML As String) As Long

C++ Syntax

int AbGetSyncInfo(BSTR* pstrXML)

See Also

ContinueSync

ContinueSyncUI

ExecuteSync

ExecuteSyncUI

AbFindFirstApp

AbFindNextApp

AbGetAppProperty

AbDestroyUI

AbGetAppPropertyW ...

AbGetAppCount

AbGetSyncLog

TxSyncCallbackFunc....

SECTION 278

AbGetSyncLog

The **AbGetSyncLog** function retrieves the contents of the synchronization log created during the TxSync process. The synchronization log is formatted as an XML document and is returned in the pstrXML parameter.

Visual Basic Syntax

Declare Function AbGetSyncLog Lib "absys" (pstrXML As String) As Boolean

C++ Syntax

bool AbGetSyncLog(BSTR* pstrXML)

See Also

ContinueSync

ContinueSyncUI

ExecuteSync

ExecuteSyncUI

AbFindFirstApp

AbFindNextApp

AbGetAppProperty

AbDestroyUI

AbGetAppPropertyW ...

AbGetAppCount

AbGetSyncInfo

TxSyncCallbackFunc

SECTION 279

TxSyncCallbackFunc

The **TxSyncCallbackFunc** is called by the TxSync process to indicate a change in the current status of the process. An application should pass the address of a TxSyncCallbackFunc as a parameter to the AbExecuteSync and AbContinueSync functions in order to retrieve notifications as the TxSync progresses. The function is passed an integer parameter that indicates the current status of the TxSync process. This functionality is not available in Embedded Visual Basic.

C++ Syntax

```
bool TxSyncCallbackFunc(int iCode)
```

See Also

ContinueSync
ContinueSyncUI
ExecuteSync
ExecuteSyncUI
AbFindFirstApp
AbFindNextApp
AbGetAppProperty
AbDestroyUI
AbGetAppPropertyW ...
AbGetAppCount
AbGetSyncInfo
AbGetSyncLog

CHAPTER 14

abUtils (System Tools)

The **abutils** library provides access to common operating system functions and device framework services. The two main objects in the library are the **File** and **Registry** objects. The **File** object can be used to perform many tasks related to files and executables and the **Registry** object is used to access the *device framework registry*.

In This Chapter

| | |
|----------------|-----|
| File | 621 |
| Registry | 635 |

File

The **File** object allows an application to easily access files stored on the client device's local file system. The **File** object provides properties and methods that can be used by an application to search the file system for files, read information from existing files and save information into new files.

The Program ID for this object is **abUtils.File**.

Methods

| |
|-----------------------|
| File::FindFirst..... |
| File::FindFirst..... |
| File::Load |
| File::Save |
| File::StartExec |
| File::StopExec |

SECTION 280**File::FindFirst**

The FindFirst method is used to search the file system for a file or directory. The FindFirst method takes two parameters. The first parameter is a string value indicating the path to be searched. The second parameter is an integer value indicating the type of search that should be performed. The possible values of this second parameter are listed in the abFileAttr enumeration. The FindFirst method returns the name of the first directory or file found if the search is successful. If no file is found an empty string is returned.

Visual Basic Syntax

```
strValue = objFile.FindFirst( strPath, intAttributes )
```

See Also

File::FindFirst.....
File::IsExecuting.....
File::Load.....
File::Save.....
File::StartExec.....
File::StopExec.....

SECTION 281**File::FindFirst**

The **FindNext** method is used to continue searching the file system for a file or directory. The **FindNext** method is used to continue a search that was started using the **FindFirst** method of the **File** object. The **FindFirst** method returns the first item matching the search criteria. After the first item has been found, the **FindNext** method can be called to search for another item that matches the criteria specified in the parameters to the **FindFirst** method. The **FindNext** method returns the name of the directory or file found if the search is successful. If no file is found an empty string is returned.

Visual Basic Syntax

```
strValue = objFile.FindNext
```

See Also

File::FindFirst.....

File::IsExecuting.....

File::Load.....

File::Save

File::StartExec.....

File::StopExec

SECTION 282**File::IsExecuting**

The **IsExecuting** property is used to determine the status of an application. The **IsExecuting** property takes one parameter. The parameter is a long value indicating the process identifier of the process to be terminated. The **IsExecuting** property returns a boolean value indicating whether the process is currently running or has been stopped. If the value is **true** the process is currently running. If the value is **false** the process is no longer running.

Visual Basic Syntax

```
boolValue = objFile.IsExecuting(lngValue)
```

Error Values

| | |
|-------------------------|--|
| ABERR_PROCESSINFO_FOUND | Unable to retrieve the status of the specified p |
|-------------------------|--|

See Also

File::FindFirst.....
File::FindFirst.....
File::Load.....
File::Save.....
File::StartExec.....
File::StopExec.....

SECTION 283**File::Load**

The **Load** method is used to read text files from the client device file system. The Load method takes one string parameter that is used to specify the full path to the text file to be opened. This method returns a string containing the entire content of the text file. The Load method can be used to read both ASCII and Unicode text files.

Visual Basic Syntax

```
strValue = objFile.Load( strPath )
```

Error Values

ABERR_READFILE_FAILED

Unable to open specified file.

See Also

File::FindFirst.....

File::FindFirst.....

File::IsExecuting.....

File::Save

File::StartExec

File::StopExec

SECTION 284

File::Save

The **Save** method can be used to create a file using text or binary information. The Save method takes two parameters. The first parameter is a string specifying the full path of the file to be created. The second parameter can be populated in two ways: when saving text information, a string containing all of the data to be saved should be passed to the second parameter; when saving binary information, a byte array of all of the data to be saved should be passed to the second parameter. When a byte array is passed to the second argument of the method, the method returns an integer value indicating the total number of bytes saved to the file.

Visual Basic Syntax

```
objFile.Save strPath, strData
```

```
intValue = objFile.Save(strPath, arrData)
```

Error Values

ABERR_TYPE_MISMATCH

An incorrect argument was passed to the meth

ABERR_SAVEFILE_FAILED

Could not create file on file system.

See Also

File::FindFirst.....

File::FindFirst.....

File::IsExecuting.....

File::Load.....

File::StartExec.....

File::StopExec.....

SECTION 285

File::StartExec

The **StartExec** method is used to launch an executable from an application. The **StartExec** method takes two string parameters. The first parameter is used to specify the full path to the executable that will be launched. The second parameter is used to pass command line arguments to the executable if necessary. The **StartExec** method returns a long value that is the process identifier of the newly launched executable.

Visual Basic Syntax

```
lngValue = objFile.StartExec(strPath, strArgs)
```

Error Values

| | |
|-------------------------|--------------------------------------|
| ABERR_EXECUTEEXE_FAILED | Unable to start specified executable |
|-------------------------|--------------------------------------|

See Also

File::FindFirst.....

File::FindFirst.....

File::IsExecuting.....

File::Load.....

File::Save

File::StopExec

SECTION 286

File::StopExec

The **StopExec** method is used to terminate a process. The **StopExec** method takes one parameter. The parameter is a long value indicating the process identifier of the process to be terminated. The **StopExec** method returns a boolean value indicating whether the process has been terminated. If the value is **true** the process has been successfully stopped.

Visual Basic Syntax

```
boolValue = objFile.StopExec(IngValue)
```

Error Values

ABERR_KILLPROCESS_FAILED Unable to kill specified process.

See Also

File::FindFirst.....

File::FindFirst.....

File::IsExecuting.....

File::Load.....

File::Save

File::StartExec

Registry

The **Registry** object can be used to access the *device framework registry*. The device framework registry is completely separate from the operating system registry and is used to store information that is only relevant to the device framework. System settings such as the unique identifier assigned to the device and the location of the system's SOAP router are stored in the device framework registry.

The Program ID for this object is **abUtils.Registry**.

Methods

Registry::ReadValue.....

SECTION 287

Registry::ReadValue

The **ReadValue** method is used to read a value from the *device framework registry*. The **ReadValue** method takes two string parameters. The first parameter indicates the path used to locate the value in the device framework registry. The second parameter is name of the value desired. Values are stored in a tree-structure in the device framework registry. Examples of paths used to retrieve values are "network\txsync" and "device". The **ReadValue** method returns the value found as a string if it is found. If the value is not found and empty string is returned.

Visual Basic Syntax

```
strValue = objRegistry.ReadValue( strPath )
```

Error Values

| | |
|--------------------------|--|
| ABERR_TYPE_MISMATCH | An incorrect argument was passed to the meth |
| ABERR_READSETTING_FAILED | Could obtain specified value from registry. |

CHAPTER 15

Enumerations

The following sections list the enumerations defined by this component library.

In This Chapter

| | |
|------------------------|-----|
| abFileAttr | 641 |
| ABHTTPSTATUS..... | 643 |
| devioTRIGGERKEY | 645 |
| devioBARCODE | 647 |
| stdioTRANSPORT | 649 |
| stdioPORT | 651 |
| stdioREADMODE..... | 653 |
| stdioFLOWCONTROL..... | 655 |
| stdioSTOPBITS..... | 657 |
| stdioPARITY | 659 |
| stdioDATABITS..... | 661 |
| stdioBAUDRATE..... | 663 |
| stdioHARDWARETYPE..... | 665 |
| stdioORIENTATION..... | 667 |
| stdioBARCODE | 669 |
| stdioTEXTFONT..... | 671 |
| stdioPRINTSIZE | 673 |

SECTION 288

abFileAttr

The file attribute values are used when searching for files or directories with the File object. The values represent the desired attributes of the file or directory to be found. To specify multiple attributes, two or more values can be combined in an **AND (&)** operation.

- abFileAttrNormal = 0
- abFileAttrReadOnly = 1
- abFileAttrHidden = 2
- abFileAttrSystem = 4
- abFileAttrVolume = 8
- abFileAttrDirectory = 16
- abFileAttrArchive = 32

- See Also

ABHTTPSTATUS.....
 devioTRIGGERKEY
 devioBARCODE
 stdioTRANSPORT
 stdioPORT
 stdioREADMODE
 stdioFLOWCONTRO
 L.....
 stdioSTOPBITS.....
 stdioPARITY.....
 stdioDATABITS.....
 stdioBAUDRATE.....
 stdioHARDWARETY
 PE.....
 stdioORIENTATION....
 stdioBARCODE
 stdioTEXTFONT.....
 stdioPRINTSIZE

SECTION 289**ABHTTPSTATUS**

The **ABHTTPSTATUS** defines the HTTP status codes returned by the Execute or PutFile methods. For more information, see RFC 2616 "<http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf>".

- ABSH_CONTINUE
- ABSH_SWITCH_PROTOCOLS
- ABSH_OK
- ABSH_CREATED
- ABSH_ACCEPTED
- ABSH_PARTIAL
- ABSH_NO_CONTENT
- ABSH_RESET_CONTENT
- ABSH_PARTIAL_CONTENT
- ABSH_AMBIGUOUS
- ABSH_MOVED
- ABSH_REDIRECT
- ABSH_REDIRECT_METHOD
- ABSH_NOT_MODIFIED
- ABSH_USE_PROXY
- ABSH_REDIRECT_KEEP_VERB
- ABSH_BAD_REQUEST
- ABSH_DENIED
- ABSH_FORBIDDEN
- ABSH_NOT_FOUND
- ABSH_BAD_METHOD
- ABSH_NONE_ACCEPTABLE
- ABSH_PROXY_AUTH_REQ
- ABSH_REQUEST_TIMEOUT
- ABSH_CONFLICT
- ABSH_GONE
- ABSH_LENGTH_REQUIRED
- ABSH_PRECOND_FAILED
- ABSH_REQUEST_TOO_LARGE
- ABSH_URI_TOO_LONG
- ABSH_UNSUPPORTED_MEDIA

- ABSH_RETRY_WITH
- ABSH_SERVER_ERROR
- ABSH_NOT_SUPPORTED
- ABSH_BAD_GATEWAY
- ABSH_SERVICE_UNAVAIL
- ABSH_GATEWAY_TIMEOUT
- ABSH_VERSION_NOT_SUP

- See Also

- abFileAttr
- devioTRIGGERKEY
- devioBARCODE
- stdioTRANSPORT
- stdioPORT
- stdioREADMODE
- stdioFLOWCONTRO
- L.....
- stdioSTOPBITS
- stdioPARITY
- stdioDATABITS
- stdioBAUDRATE
- stdioHARDWARETY
- PE.....
- stdioORIENTATION....
- stdioBARCODE
- stdioTEXTFONT
- stdioPRINTSIZE

SECTION 290

devioTRIGGERKEY

The **devioTRIGGERKEY** enumeration defines which Windows CE application launch key will be used to generate a soft trigger with the Scanner object. For more information about *application button mappings* refer to the Windows CE or vendor documentation.

- **devioTRIGGERKEY_NONE** = 0
- **devioTRIGGERKEY_APPLAUNCH1** = 0xC1
- **devioTRIGGERKEY_APPLAUNCH2** = 0xC2
- **devioTRIGGERKEY_APPLAUNCH3** = 0xC3
- **devioTRIGGERKEY_APPLAUNCH4** = 0xC4
- **devioTRIGGERKEY_APPLAUNCH5** = 0xC5
- **devioTRIGGERKEY_APPLAUNCH6** = 0xC6
- **devioTRIGGERKEY_APPLAUNCH7** = 0xC7
- **devioTRIGGERKEY_APPLAUNCH8** = 0xC8
- **devioTRIGGERKEY_APPLAUNCH9** = 0xC9
- **devioTRIGGERKEY_APPLAUNCH10** = 0xCA
- **devioTRIGGERKEY_APPLAUNCH11** = 0xCB
- **devioTRIGGERKEY_APPLAUNCH12** = 0xCC
- **devioTRIGGERKEY_APPLAUNCH13** = 0xCD
- **devioTRIGGERKEY_APPLAUNCH14** = 0xCE
- **devioTRIGGERKEY_APPLAUNCH15** = 0xCF

See Also

abFileAttr
 ABHTTPSTATUS.....
 devioBARCODE
 stdioTRANSPORT
 stdioPORT.....
 stdioREADMODE.....
 stdioFLOWCONTRO
 L.....
 stdioSTOPBITS.....
 stdioPARITY.....
 stdioDATABITS.....
 stdioBAUDRATE.....
 stdioHARDWARETY
 PE
 stdioORIENTATION....

studioBARCODE
studioTEXTFONT.....
studioPRINTSIZE

SECTION 291

devioBARCODE

The **devioBARCODE** enumeration defines the barcode symbologies that are supported by the `Scanner` object.

- `devioBARCODE_NONE = 0`
- `devioBARCODE_CODE39 = 1`
- `devioBARCODE_CODE93 = 2`
- `devioBARCODE_CODE49 = 4`
- `devioBARCODE_I2OF5 = 8`
- `devioBARCODE_D2OF5 = 16`
- `devioBARCODE_CODABAR = 32`
- `devioBARCODE_UPC = 64`
- `devioBARCODE_UPCE0 = 128`
- `devioBARCODE_UPCE1 = 256`
- `devioBARCODE_UPCA = 512`
- `devioBARCODE_CODE128 = 1024`
- `devioBARCODE_CODE16K = 2048`
- `devioBARCODE_PLESSEY = 4096`
- `devioBARCODE_CODE11 = 8192`
- `devioBARCODE_MSI = 16384`
- `devioBARCODE_PDF417 = 32768`
- `devioBARCODE_EAN13 = 65536`
- `devioBARCODE_EAN8 = 131072`
- `devioBARCODE_TRIOPTIC39 = 262144`
- `devioBARCODE_ALL = 0xFFFFFFFF`

- See Also

- `abFileAttr`
- `ABHTTPSTATUS`
- `devioTRIGGERKEY`
- `stdioTRANSPORT`
- `stdioPORT`
- `stdioREADMODE`
- `stdioFLOWCONTRO`
- `L`
- `stdioSTOPBITS`
- `stdioPARITY`
- `stdioDATABITS`
- `stdioBAUDRATE`
- `stdioHARDWARETY`

PE.....
stdioORIENTATION....
stdioBARCODE
stdioTEXTFONT
stdioPRINTSIZE

SECTION 292

stdioTRANSPORT

The **stdioTRANSPORT** enumeration defines which transport will be used to communicate with the Printer .

- **stdioTRANSPORT_SERIAL** = 1
- **stdioTRANSPORT_IRDA** = 2

- See Also

- abFileAttr
- ABHTTPSTATUS.....
- devioTRIGGERKEY
- devioBARCODE
- stdioPORT
- stdioREADMODE
- stdioFLOWCONTRO
- L.....
- stdioSTOPBITS.....
- stdioPARITY
- stdioDATABITS.....
- stdioBAUDRATE.....
- stdioHARDWARETY
- PE.....
- stdioORIENTATION....
- stdioBARCODE
- stdioTEXTFONT
- stdioPRINTSIZE

SECTION 293

stdioPORT

The **stdioPORT** enumeration defines which port will be used with the Serial control.

- **stdioPORT_ONE** = 1
- **stdioPORT_TWO** = 2
- **stdioPORT_THREE** = 3
- **stdioPORT_FOUR** = 4
- **stdioPORT_FIVE** = 5
- **stdioPORT_SIX** = 6
- **stdioPORT_SEVEN** = 7
- **stdioPORT_EIGHT** = 8

- See Also

abFileAttr
ABHTTPSTATUS.....
devioTRIGGERKEY
devioBARCODE
stdioTRANSPORT
stdioREADMODE.....
stdioFLOWCONTRO
L.....
stdioSTOPBITS.....
stdioPARITY.....
stdioDATABITS.....
stdioBAUDRATE.....
stdioHARDWARETY
PE.....
stdioORIENTATION....
stdioBARCODE
stdioTEXTFONT
stdioPRINTSIZE

SECTION 294

stdioREADMODE

The **stdioREADMODE** enumeration defines if a read operation will be synchronous (blocking) or asynchronous (non-blocking).

- **stdioREADMODE_SYNCHRONOUS** = 0
- **stdioREADMODE_ASYNCHRONOUS** = 1

- See Also

- **abFileAttr**
- **ABHTTPSTATUS**
- **devioTRIGGERKEY**
- **devioBARCODE**
- **stdioTRANSPORT**
- **stdioPORT**
- **stdioFLOWCONTRO**
L
- **stdioSTOPBITS**
- **stdioPARITY**
- **stdioDATABITS**
- **stdioBAUDRATE**
- **stdioHARDWARETY**
PE
- **stdioORIENTATION**....
- **stdioBARCODE**
- **stdioTEXTFONT**
- **stdioPRINTSIZE**

SECTION 295

stdioFLOWCONTROL

The **stdioFLOWCONTROL** enumeration defines which flow control will be used with the Serial control.

- **stdioFLOWCONTROL_NONE** = 0
- **stdioFLOWCONTROL_XONXOFF** = 1
- **stdioFLOWCONTROL_RTSCTS** = 2
- **stdioFLOWCONTROL_BOTH** = 3

- See Also

abFileAttr
ABHTTPSTATUS.....
devioTRIGGERKEY
devioBARCODE
stdioTRANSPORT
stdioPORT
stdioREADMODE.....
stdioSTOPBITS.....
stdioPARITY
stdioDATABITS.....
stdioBAUDRATE.....
stdioHARDWARETY
PE.....
stdioORIENTATION....
stdioBARCODE
stdioTEXTFONT
stdioPRINTSIZE

SECTION 296

stdioSTOPBITS

The **stdioSTOPBITS** enumeration defines the stopbits used with the Serial control.

- **stdioSTOPBITS_ONE** = 1
- **stdioSTOPBITS_TWO** = 2

- See Also

abFileAttr
ABHTTPSTATUS.....
devioTRIGGERKEY
devioBARCODE
stdioTRANSPORT
stdioPORT
stdioREADMODE.....
stdioFLOWCONTRO
L.....
stdioPARITY
stdioDATABITS.....
stdioBAUDRATE.....
stdioHARDWARETY
PE.....
stdioORIENTATION....
stdioBARCODE
stdioTEXTFONT.....
stdioPRINTSIZE

SECTION 297

stdioPARITY

The **stdioPARITY** enumeration defines the parity used with the Serial control.

- **stdioPARITY_NONE** = 0
- **stdioPARITY_ODD** = 1
- **stdioPARITY_EVEN** = 2

- See Also

abFileAttr
ABHTTPSTATUS.....
devioTRIGGERKEY
devioBARCODE
stdioTRANSPORT
stdioPORT.....
stdioREADMODE.....
stdioFLOWCONTRO
L.....
stdioSTOPBITS.....
stdioDATABITS.....
stdioBAUDRATE.....
stdioHARDWARETY
PE.....
stdioORIENTATION....
stdioBARCODE
stdioTEXTFONT
stdioPRINTSIZE

SECTION 298

stdioDATABITS

The **stdioDATABITS** enumeration defines the databits used with the Serial control.

- **stdioDATABITS_4** = 4
- **stdioDATABITS_5** = 5
- **stdioDATABITS_6** = 6
- **stdioDATABITS_7** = 7
- **stdioDATABITS_8** = 8

- See Also

abFileAttr
ABHTTPSTATUS.....
devioTRIGGERKEY
devioBARCODE
stdioTRANSPORT
stdioPORT.....
stdioREADMODE
stdioFLOWCONTRO
L.....
stdioSTOPBITS.....
stdioPARITY.....
stdioBAUDRATE.....
stdioHARDWARETY
PE.....
stdioORIENTATION....
stdioBARCODE
stdioTEXTFONT.....
stdioPRINTSIZE

SECTION 299

stdioBAUDRATE

The **stdioBAUDRATE** enumeration defines which baudrate used with the Serial control.

- **stdioBAUDRATE_1200** = 1
- **stdioBAUDRATE_2400** = 2
- **stdioBAUDRATE_4800** = 3
- **stdioBAUDRATE_9600** = 4
- **stdioBAUDRATE_19200** = 5
- **stdioBAUDRATE_38400** = 6
- **stdioBAUDRATE_57600** = 7
- **stdioBAUDRATE_115200** = 8

▪ See Also

abFileAttr
 ABHTTPSTATUS.....
 devioTRIGGERKEY
 devioBARCODE
 stdioTRANSPORT
 stdioPORT
 stdioREADMODE.....
 stdioFLOWCONTRO
 L.....
 stdioSTOPBITS.....
 stdioPARITY.....
 stdioDATABITS.....
 stdioHARDWARETY
 PE.....
 stdioORIENTATION....
 stdioBARCODE
 stdioTEXTFONT
 stdioPRINTSIZE

SECTION 300

stdioHARDWARETYPE

The **stdioHARDWARETYPE** enumeration defines which built-in hardware or peripherals can be accessed through the Printer control.

- **stdioHARDWARETYPE_SCANNER** = 0
- **stdioHARDWARETYPE_MAGNETICCARDREADER** = 1
- **stdioHARDWARETYPE_SERIAL** = 2
- **stdioHARDWARETYPE_PRINTER** = 3

See Also

abFileAttr
ABHTTPSTATUS.....
devioTRIGGERKEY
devioBARCODE
stdioTRANSPORT
stdioPORT
stdioREADMODE
stdioFLOWCONTRO
L.....
stdioSTOPBITS.....
stdioPARITY.....
stdioDATABITS.....
stdioBAUDRATE.....
stdioORIENTATION....
stdioBARCODE
stdioTEXTFONT
stdioPRINTSIZE

SECTION 301

stdioORIENTATION

The **stdioORIENTATION** enumeration defines the orientation of the text or barcode printed with the Printer control.

- **stdioORIENTATION_HORIZONTAL = 0**
- **stdioORIENTATION_VERTICAL = 1**

- See Also

- **abFileAttr**
- **ABHTTPSTATUS**
- **devioTRIGGERKEY**
- **devioBARCODE**
- **stdioTRANSPORT**
- **stdioPORT**
- **stdioREADMODE**
- **stdioFLOWCONTRO**
- **L**
- **stdioSTOPBITS**
- **stdioPARITY**
- **stdioDATABITS**
- **stdioBAUDRATE**
- **stdioHARDWARETY**
- **PE**
- **stdioBARCODE**
- **stdioTEXTFONT**
- **stdioPRINTSIZE**

SECTION 302

stdioBARCODE

The **stdioBARCODE** enumeration defines which barcode will be printed with the Printer control.

- **stdioBARCODE_CODE128** = 0
- **stdioBARCODE_CODE39** = 1
- **stdioBARCODE_CODE93** = 2
- **stdioBARCODE_CODABAR** = 3
- **stdioBARCODE_EAN128** = 4
- **stdioBARCODE_I2OF5** = 5
- **stdioBARCODE_MAXICODE** = 6
- **stdioBARCODE_PDF417** = 7
- **stdioBARCODE_POSTNET** = 8
- **stdioBARCODE_UPCA** = 9

See Also

abFileAttr
 ABHTTPSTATUS.....
 devioTRIGGERKEY
 devioBARCODE
 stdioTRANSPORT
 stdioPORT
 stdioREADMODE
 stdioFLOWCONTRO
 L.....
 stdioSTOPBITS.....
 stdioPARITY.....
 stdioDATABITS.....
 stdioBAUDRATE.....
 stdioHARDWARETY
 PE
 stdioORIENTATION....
 stdioTEXTFONT.....
 stdioPRINTSIZE

SECTION 303

stdioTEXTFONT

The **stdioTEXTFONT** enumeration defines which font will be used to print text with the Printer control.

- **stdioTEXTFONT_DEFAULT** = 0
- **stdioTEXTFONT_ONE** = 1
- **stdioTEXTFONT_TWO** = 2
- **stdioTEXTFONT_THREE** = 3

See Also

abFileAttr
ABHTTPSTATUS.....
devioTRIGGERKEY....
devioBARCODE
stdioTRANSPORT
stdioPORT.....
stdioREADMODE.....
stdioFLOWCONTRO
L.....
stdioSTOPBITS.....
stdioPARITY.....
stdioDATABITS.....
stdioBAUDRATE.....
stdioHARDWARETY
PE
stdioORIENTATION....
stdioBARCODE
stdioPRINTSIZE

SECTION 304

stdioPRINTSIZE

The **stdioPRINTSIZE** enumeration defines which font size will be used to print a barcode or text with the Printer control.

- **stdioPRINTSIZE_SMALL** = 0
- **stdioPRINTSIZE_DEFAULT** = 1
- **stdioPRINTSIZE_MEDIUM** = 2
- **stdioPRINTSIZE_LARGE** = 3

See Also

abFileAttr
ABHTTPSTATUS.....
devioTRIGGERKEY....
devioBARCODE
stdioTRANSPORT
stdioPORT.....
stdioREADMODE.....
stdioFLOWCONTRO
L.....
stdioSTOPBITS.....
stdioPARITY.....
stdioDATABITS.....
stdioBAUDRATE.....
stdioHARDWARETY
PE
stdioORIENTATION....
stdioBARCODE
stdioTEXTFONT.....

CHAPTER 16

Java Device Framework

CHAPTER 17

Client Workers

Many tasks are completed during the execution of an Atoma™ Tx-Sync. For example, monitoring information is collected and transferred, configuration information is retrieved and queued messages are transmitted. However, there may be tasks that are closely tied with a particular application or industry that are not completed during a standard Tx-Sync operation. Developers are empowered to include these tasks during each Tx-Sync execution using **client workers**.

A **client worker** is a client device application that is executed during Tx-Sync execution. On Windows Powered clients, a client worker can be an executable file (.EXE), an ActiveX Object, or an eMbedded Visual Basic application.

Client workers fall into two categories: **pre-workers** and **post-workers**. A pre-worker is an application that is called during the data preparation stage of the Tx-Sync process. As a result, pre-workers can be used to create and/or prepare files that are transmitted to the server during the synchronization. A post-worker is an application that is called during the package processing stage of the Tx-Sync process. Post-workers can be used to read information that has been returned to the client from the Tx-Sync server.



In This Chapter

| | |
|----------------------------------|-----|
| Developing a client worker | 679 |
| Deploying a client worker | 681 |

SECTION 305

Developing a client worker

A Client Worker can be developed in three ways. It can be developed as an Embedded Visual Basic application, a C\C++ executable or a C\C++ dynamic link library (DLL) that exposes a COM interface. While any task can be performed by a client worker, one common task is the transmission and retrieval of files between the client device and synchronization server.

The Tx-Sync agent uses two folders, the *inbox* and *outbox*, to manage the transmission of files to and from a server. A client worker can utilize the folders to send and receive files from a server if necessary. Pre-workers may send a file and or folder by placing it in the outbox folder at the time they are executed. The files and folders in the client outbox are transmitted to the server and deployed to the corresponding inbox on the server. Post-workers may read files received from a server by locating the files within the inbox at the time they are executed. The *inbox* and *outbox* are cleaned during each synchronization. Regular client applications (i.e. other than client workers) can access the inbox and outbox at anytime however it is generally best to use a client worker when writing or reading to these locations.

The full paths to the *inbox* and *outbox* are passed to client workers when they are launched. Client Workers are also provided with a *type* parameter when they are activated. This *type* parameter indicates whether the client worker has been launched as a pre-worker or a post-worker. The *type* parameter is an integer. If the value is zero (0) the worker is being launched as a pre-worker. If the value is one (1) the worker is being launched as a post-worker.

Executables

When a client worker is implemented as an executable, it is called from the command line by the Tx-Sync Agent. Command line arguments are passed to the client worker executable in the following format:

```
-i InboxPath -o OutboxPath -t WorkerType
```

COM objects

When a client worker is implemented as a COM object, the client worker is created during the Tx-Sync process and the following method is called:

```
DoSync(BSTR inbox, BSTR outbox, integer type)
```

This method must be implemented in the client worker COM object.

Embedded Visual Basic

When a client worker is implemented as an Embedded Visual Basic application the application is launched from the command line by the Tx-Sync Agent. No command line arguments are passed to Embedded Visual Basic applications.

User Interface Guidelines

A client worker should not have a user interface that requires user input. During Tx-Sync, the Tx-Sync agent displays messages and status indicators to the user. If a client worker covers the information displayed, the user may become alarmed or may be unable to see important information regarding the overall process. Also note that the overall Tx-Sync process will not proceed to the next stage until each client-worker has completed. If a client-worker fails to complete because of a lingering message or window, the overall Tx-Sync process will be suspended until the client-worker is closed completely.

Error Handling

The overall Tx-Sync process will not abort if a client worker aborts. Since client worker applications may or may not be present during a Tx-Sync operation, the Tx-Sync Agent does not abort the synchronization process if an error occurs when attempting to execute a client worker.

See Also

Deploying a client
worker

SECTION 306

Deploying a client worker

Two steps must be performed in order to deploy a client worker. The first step is to create an application that includes client worker files using the application deployment tool on the administration Console. The second step is to register this application as a client worker through the administration Console after it has been added as an application. As with any application on the system, the client worker application must be assigned to an application group before it will be deployed to a device. If needed, the client worker application can be assigned to multiple groups. Note that if a client device synchronizes with a group that is not associated with the client worker application, the client worker application will be removed from the device if present and subsequently will not be run on the device.

To remove a client worker, reverse the steps performed to deploy the client worker. Remove the client worker from any groups where it is assigned. Then, if you wish to completely delete the worker application, remove the application from the set of registered applications.

See Also

Developing a client
worker

CHAPTER 18

Server-Side Development

Many mobile solutions require development on two platforms: a client application that runs on the mobile device and a server application(s) that provides or processes data for the client application. The Atoma system allows mobile solution developers to integrate their applications into the Tx-Sync process performed by each client device. The methods of performing this integration are defined in this chapter.

The system can be highly customized to perform almost any task imaginable during the Tx-Sync process. Integration with the Tx-Sync process allows developers to ensure that critical tasks are reliably performed and to take advantage of the services provided by the process such as binary file transfer, compressed data communication and client device database management (data piping).

In This Chapter

| | |
|----------------------------------|-----|
| Data Filters | 685 |
| Synchronization Guard | 691 |
| Server Workers | 693 |
| Synchronization Parameters | 697 |
| Java Mail Utility | 699 |

CHAPTER 19

Data Filters

A **Data Filter** is a web service that is called during the Tx-Sync process to aid in determining the database records that are transferred to a client device during the data piping process. Data Filters work in conjunction with Data Packages to identify the records that are sent to a device.

In This Chapter

| | |
|----------------------------|-----|
| Data Packages..... | 687 |
| Data Filter Interface..... | 689 |

SECTION 307

Data Packages

A **Data Package** is essentially a structural definition of a client device database table. A Data Package defines the name of the table, the name of the columns in the table, the datatype of each column, and the source tables and columns from a central database server that correspond to the columns of the table on the client device. Primary keys and indexes on the client device table can also be defined in a Data Package.

When a Data Filter is deployed it is always associated with a Data Package. When the Data Filter is created it is usually necessary to know the names of the source tables used in the package and the names of the columns of each source table.

See Also

Data Filter Interface.....

SECTION 308

Data Filter Interface

A **Data Filter** is a web service that exposes two methods. These two methods make up the *Data Filter Interface*. The two methods are defined as follows:

```
String applyFilter(String packageId, String dataKey)
String test(String testParam)
```

When a Data Filter is written in a language that is case sensitive, the spelling an case of the Data Filter methods and parameters should be identical to the what is listed above.

ApplyFilter Method

The Applyfilter method is called during the Tx-Sync process to execute a Data Filter. The *packageId* parameter contains the name of the Data Package being processed. This parameter might be used to validate that the filter being called is compatible with the Data Package to which it is assigned. The *dataKey* parameter contains the synchronization parameters entered by the user during the Tx-Sync process. The synchronization parameters are passed to the Data Filter in the form of an XML document that contains the name of each parameter and the value entered by the device user. (See Synchronization Parameters)

The *dataKey* parameter is usually very useful in a Data Filter. The values passed through the dataKey parameter can be used to determine the records retrieved from the source database and inserted into the client database.

The string returned by the ApplyFilter method is a Structured Query Language (SQL) *where*-clause (the word *where* should not be included in the string). The where-clause conditions returned by the ApplyFilter method will be added to an SQL *select* statement that is automatically generated from the client table structure defined in the associated Data Package. The resulting SQL statement will be executed on the source database to retrieve the data records desired on the client device database.

Test Method

The Test method is called when the *Test Data Filter* option is selected from an administrative console. The *testParam* value is entered by the person requesting the test and the string value returned by the method is displayed to the person requesting the test when the function returns. This method is solely used for administrative purposes. It can be used by a system administrator to determine whether a Data Filter has been deployed correctly and also to ensure that the system server can communicate with the deployed Data Filter.

See Also

Data Packages.....

SECTION 309

Synchronization Guard

The **Synchronization Guard** is a web service that can be used to perform application or site specific tasks before any Server Workers are executed during the Tx-Sync process. The Synchronization Guard must expose a method that allows the server to call the guard before each device synchronization is started. The following method must be implemented:

integer validate(String deviceID, String syncParam)

The *validate* method will be called when each client device attempts to perform a synchronization. The unique identifier assigned to the device will be passed in the *deviceID* parameter. The synchronization parameters entered by the device user will be passed in the *syncParam* parameter (See also Synchronization Parameters). The synchronization parameters can be used by the Synchronization Guard to determine whether the client device will be allowed to continue with the synchronization.

The integer value returned by the *validate* method is used by the system indicate whether the device will be allowed to continue with the synchronization. A value of 1 indicates that the client device will be allowed to continue with the synchronization. Any value other than 1 indicates that the synchronization should be rejected.

See Also

Data Filters
Server Workers
Synchronization
Parameters
Java Mail Utility

CHAPTER 20

Server Workers

A **Server Worker** is a Java[®] class that is called during the Tx-Sync process to perform any desired task. By default the Tx-Sync process performs several general system tasks. A Server Worker can be used to include additional tasks that may be more application specific whenever a synchronization task is performed.

A Server Worker allows the Tx-Sync process to be extended with almost complete freedom. However care should be taken when adding new Server Workers to the synchronization process. Depending on the error options selected on an installation, a synchronization might fail or abort if a Server Worker raises an error.

One of the primary uses of a server worker is to transmit, receive and process application related files from a client device. When a client device synchronizes any files transmitted from the device are placed in an *inbox* directory on the server. The path to this directory is supplied to a Server Worker when it is executed, allowing the Server Worker to read and/or process any files sent from a client device. The path to the client device's *outbox* directory is also supplied to the Server Worker. Any files placed in the *outbox* directory by the Server Worker will be sent to the client device during the synchronization. These files will appear in an *inbox* folder on the client device (see also Client Workers).

In This Chapter

Server Worker Interface.....695

SECTION 310

Server Worker Interface

A **Server Worker** is a class that extends the class **SyncWorker** in the *com.abacoinc.xoom.syncsvr.worker* package. The **SyncWorker** class provides default implementations for many of the methods used by the Atoma server to manage a **Server Worker** however a number of methods must be implemented in the **Server Worker** class. A **Server Worker** must implement the following methods:

```
public String getWorkerID()
protected void executeWorker(Message message) throws SyncException
```

The class **SyncException** is part of the *com.abacoinc.xoom.syncsvr* package. Additionally a **Server Worker** may implement/override the following methods:

```
public void create() throws PoolException {}
public void remove() throws PoolException {}
public void activate() throws PoolException {}
public void passivate() throws PoolException {}
```

The class **PoolException** is part of the *com.abacoinc.utils.pool* package.

GetWorkerID Method

The **GetWorkerID** method is used by the system server to uniquely identify a **ServerWorker**. This method should return a global unique identifier as a string. The administration console of the system provides a GUID generator that can be used to create such an identifier. Once a unique identifier has been obtained it should be returned by this method every time it is called.

ExecuteWorker Method

The **ExecuteWorker** method is called during the Tx-Sync process to start the **Server Worker**. Inside the **ExecuteWorker** method the **Server Worker** can perform any tasks that it wishes to complete during each device synchronization. The *message* parameter is a collection of string data values associated with the synchronization being performed. The **Message** object has a method called *getProperty* that can be used to retrieve any of the values passed in the *message* parameter. The key identifying the value is passed as a string to the *getProperty* method and the string data value corresponding to the key is returned by the method. The data values available during the Tx-Sync process are defined as follows:

| Key | Value |
|----------|---|
| syncID | unique identifier assigned to a synchronization |
| deviceID | unique identifier assigned to client device performing synchronization |
| inbox | the full path to the client device's inbox; contains files sent from client |

| Key | Value |
|------------|--|
| outbox | the full path to the client device's outbox; files placed here are sent to client |
| syncParams | the synchronization parameters entered by the user in XML format; see Synchronization Parameters |
| onError | flag indicating how errors should be handled; possible values are: ABORT - error should be thrown in <i>executeWorker</i> method ABORTANDMAIL - error should be thrown in <i>executeWorker</i> method and an e-mail should be sent using Email utility CONTINUE - error should not be thrown in <i>executeWorker</i> method CONTINUEANDMAIL - error should not be thrown in <i>executeWorker</i> method but an e-mail should be sent using Email utility See also Java Mail Utility |
| groupID | unique identifier assigned to group selected by user during synchronization |

Create Method

The Create method is used by the server's pool manager. The Create method will be called once during the lifecycle of a Server Worker when it is first instantiated. This method can be used to perform initialization tasks required by the Server Worker.

Remove Method

The Remove method is used by the server's pool manager. The Remove method will be called once during the lifecycle of a Server Worker when it is finally destroyed. This method can be used to ensure that all resources are destroyed and any desired clean-up tasks are performed when a Server Worker is destroyed.

Activate Method

The Activate method is used by the server's pool manager. The Activate method is called when the server pool manager makes the Server Worker available for usage. This method can be used to create resources that will be needed when the *executeWorker* method is called. This method will be called one or more times in the lifecycle of the Server Worker.

Passivate Method

The Passivate method is used by the server's pool manager. The Passivate method is called when the server pool manager *idles* the Server Worker. A Server Worker is *idled* when it will not be used for an extended period. This method can be used to release resources that will not be used while the Server Worker is idle. This method will be called one or more times in the lifecycle of the Server Worker.

SECTION 311

Synchronization Parameters

Synchronization Parameters are values entered by a device user during the Tx-Sync process. The values entered depend on the Group selected during the Tx-Sync process. The parameters associated with each Group are defined in the administrative console of the system.

The Synchronization Parameters are passed to many server objects such as Server Workers and Data Filters to allow these objects to customize the Tx-Sync process depending on values entered by the user. These values are passed to the server objects in a XML document similar to the following:

```
<syncparams id="0000bdf10000baea000000e95e5fcc16"
name="MyGroup">
  <param
name="id">123456</param>
  <param name="psswd">Xc783k</param>
</syncparams>
```

| | |
|-----------------------|---|
| "syncparams" node | root node of XML document; document element |
| "groupid" attribute | unique identifier associated with group chosen by device user |
| "groupname" attribute | name assigned to group chosen by device user |
| "param" node | node containing parameter information; one "param" node will appear in the document for each synchronization parameter defined for the group; node value is the data entered by the device user for the parameter |
| "name" attribute | name of synchronization parameter |

For objects that share the Java™ classpath used by the system server, a synchronization parameter utility is provided in the class *com.abacoinc.xoom.utils.SyncParamsHandler*. This class provides two methods *readParams* and *getParams* that can be used to simplify the process of parsing the XML document containing the synchronization parameters. The *readParams* method is passed one string parameter containing the XML representation of the parameters. This method returns a boolean value indicating whether the document has been successfully parsed. After the *readParams* method has been called, the *getParams* method can be called to retrieve a *java.util.HashMap*. The *getParams* does not take any parameters. The values entered by the user can be retrieved on demand from the *HashMap* using the *get* method and supplying the name of the desired parameter as the key.

```
import com.abacoinc.xoom.utils.SyncParamsHandler;

// create a new SyncParamsHandler
SyncParamsHandler handler = new
SyncParamsHandler();
String syncParams = new String();

// obtain the syncparams document.....
// syncParams = "<syncparams ....."

//parse document
if (handler.readParams(syncParams)) {

    java.util.HashMap params = handler.getParams();

    String myParam = (String) params.get("MyKey");

}
```

See Also

Data Filters.....
Synchronization Guard .
Server Workers.....
Java Mail Utility

SECTION 312

Java Mail Utility

An e-mail utility is provided in the class *com.abacoinc.xoom.utils.Email* for classes that share the system's classpath on a server. This class provides a static method *send* that can be used to easily send an e-mail message. The *send* method can be called using either of the following definitions:

`void send(String to, String from, String host, String subject, String message)` throws *MessagingException*

`void send(String groupID, String subject, String message)` throws *MessagingException*, *SQLException*

The *subject* parameter is used to specify the subject line of the e-mail message. The *message* parameter is used to specify the body of the e-mail message. The *to* parameter specifies the e-mail address of the message recipient. The *from* parameter specifies the e-mail address of the message sender. The *host* parameter specifies the network address of the POP3 e-mail server that will be used to transmit the message.

The *to*, *from*, and *host* parameters can be replaced by a *groupID* parameter that is used to specify the unique identifier of a valid application group on the system server. When the *groupID* parameter is specified, the e-mail message will be sent to the user configured as the contact person responsible for the group and the server configuration settings will be used to obtain the e-mail server used to transmit the message.

```
import com.abacoinc.xoom.utils.Email;

public class TestMail {

    public static void main(String[] args) {

        // send a test message
        Email.send( "myfriend@theirdomain.com",
                   "me@mydomain.com",
                   "mailserver.mydomain.com",
                   "A test message",
                   "This is a test message. Please do not
respond!" );

    }

}
```

See Also

Data Filters
Synchronization Guard .
Server Workers
Synchronization
Parameters

CHAPTER 21

Samples

In This Chapter

| | |
|--|-----|
| Storing information between sessions | 703 |
| Reading credit cards | 705 |
| Using the serial port..... | 707 |
| Printing a barcode..... | 709 |
| Using the scanner control..... | 711 |
| Receiving power notifications | 713 |
| Enumerating connections | 715 |
| Dialing up a remote site | 717 |
| Calling a BAPI | 719 |
| Posting an IDoc | 723 |
| Queueing soap requests..... | 727 |
| Ping | 731 |
| Making SOAP calls | 733 |

SECTION 313

Storing information between sessions

```

'declare application object
Dim Application as ABASPEX.Application

'create application object
Set Application = CreateObject("ABASPEX.Application")

'access application store or create a new one
Application.AppName = "VirtualDirectory"

'synchronize access to the application store
Application.Lock

'update some information
If (Application("Visits") <> Empty) Then
    Application("Visits") = Application("Visits") + 1
Else
    Application("Visits") = 1
End If

'unlock the application store
Application.Unlock

'destroy the application object
Set Application = Nothing

```

See Also

- Reading credit cards
- Using the serial port.....
- Printing a barcode.....
- Using the scanner
- control
- Receiving power
- notifications.....
- Enumerating
- connections.....
- Dialing up a remote
- site.....
- Calling a BAPI
- Posting an IDoc
- Queueing soap
- requests
- Ping.....

Making SOAP calls

SECTION 314

Reading credit cards

```
'declare msr object
Dim Msr as ABDEVIO.Msr

Public Sub InitMSR()
    'create msr object with event support
    Set Msr =
CreateObjectWithEvents("Abdevio.Msr", "Msr_")
    'check if msr is supported. To use the msr
attached to a
    'printer, use the HardwareRead method of the
printer object.
    If Msr.Supported Then
        'enable msr
        Msr.Enabled = True
    End If
End Sub

Public Sub DestroyMSR()
    'destroy msr object
    Set Msr = Nothing
End Sub
```

```
'Msr event sink
Public Sub Msr_OnDataAvailable()
    'display msr data
    MsgBox "Data = " & Msr.GetData
End Sub
```

See Also

- Storing information
between sessions.....
- Using the serial port.....
- Printing a barcode.....
- Using the scanner
control.....
- Receiving power
notifications.....
- Enumerating
connections.....
- Dialing up a remote
site.....
- Calling a BAPI
- Posting an IDoc
- Queueing soap
requests
- Ping.....
- Making SOAP calls

SECTION 315

Using the serial port

```
Public Sub ReadSerial()  
    Dim oSerial as ABSTDIO.Serial  
    Dim sData as String  
    'create serial object without event support  
    Set oSerial = CreateObject("abstdio.Serial")  
  
    'set serial port properties  
    oSerial.baudrate = stdioBAUDRATE_19200  
    oSerial.Port = Comb1.ListIndex + 1  
  
    'open serial port  
    oSerial.Enabled = True  
  
    'write to the serial port  
    oSerial.Write "hello World!!!"  
  
    'blocking read from the serial port (10  
seconds)  
    'remember that the serial port also supports  
the  
    'OnDataAvailable event if asynchronous reading  
is  
    'required.  
    sData =  
oSerial.Read(stdioREADMODE_SYNCHRONOUS,10)  
    If Len(sData) > 0 Then  
        MsgBox "Data = " & sData  
    End If  
  
    'destroy serial object  
    Set oSerial = Nothing  
End Sub
```

The following example shows an asynchronous read of the serial port. When data is read by the serial port, the OnDataAvailable event is triggered where the GetData method of the serial object can be used to retrieve the information that has been read.

Dim oSerial As Object

Public Sub InitSerial()

 Set oSerial = CreateObjectWithEvents("abstdio.Serial", "Serial_")

End Sub

Public Sub DestroySerial()

 Set oSerial = Nothing

End Sub

Public Sub Serial_OnDataAvailable()

 MsgBox "Data = " & Serial.GetData

End Sub

See Also

Storing information
between sessions.....
Reading credit cards
Printing a barcode.....
Using the scanner
control.....
Receiving power
notifications.....
Enumerating
connections.....
Dialing up a remote
site.....
Calling a BAPI
Posting an IDoc
Queueing soap
requests
Ping.....
Making SOAP calls

SECTION 316

Printing a barcode

```
Private Sub MyPrint()  
    'declare printer object  
    dim Printer as ABSTDIO.Printer  
  
    'create printer object  
    Set Printer = CreateObject("abstdio.Printer")  
    'select printer  
    Printer.PrinterType = "Cameo2"  
  
    'select transport. properties like baudrate,  
    'parity and databits are set to the default  
    'parameters of the selected printer  
  
    Printer.Transport = stdioTRANSPORT_SERIAL  
  
    'format label  
    Printer.StartDoc  
    Printer.PrintSize = stdioPRINTSIZE_LARGE  
    Printer.PrintText "Invoice"  
    Printer.DrawLine , , 350, -1  
    Printer.PrintSize = stdioPRINTSIZE_DEFAULT  
    Printer.Advance 1  
    Printer.PrintText "Job No: 1"  
    Printer.PrintText "Customer: Joe Boggs"  
    Printer.PrintText "Address: 1234 St. Place"  
    Printer.PrintText "Phone: 123-456-7890"  
    Printer.Advance 1  
    Printer.DrawLine , , 350, -1  
    Printer.Advance 1  
    Printer.PrintText "Total: $100"  
    Printer.DrawLine , , 350, -1  
    Printer.Advance 1  
    Printer.PrintBarcode stdioBARCODE_CODE39, "12345678"  
    Printer.Advance 3  
    Printer.EndDoc  
  
    'destroy printer object  
    Set Printer = Nothing  
End Sub
```

See Also

Storing information

between sessions.....
Reading credit cards
Using the serial port.....
Using the scanner
control
Receiving power
notifications
Enumerating
connections.....
Dialing up a remote
site.....
Calling a BAPI
Posting an IDoc
Queueing soap
requests
Ping.....
Making SOAP calls

SECTION 317

Using the scanner control

```
'declare scanner object
Dim Scanner as ABDEVIO.Scanner

Public Sub InitScanner()
    'create scanner object with event support
    Set Scanner =
CreateObjectWithEvents("Abdevio.Scanner",
"Scanner_")

    'check if scanner is supported
    If Scanner.Supported Then
        'enable the scanner
        Scanner.Enabled = True
    End If
End Sub

Public Sub ReadFromScanner()
    'perform a soft trigger - activates the
scanner laser
    'as if the scanner had been activated manually
    Scanner.SoftTrigger
End Sub

Public Sub
    'destroy scanner object
    Set Scanner = Nothing
End Sub
```



```
'Scanner event sink
Public Sub Scanner_OnDataAvailable()
    'display scanner data
    MsgBox "Data = " & Scanner.GetData
    'display scanner symbologies -
    'BarcodePrint is included in the mABDEVIO.bas module
    MsgBox " Symbology = " & BarcodePrint(Scanner.GetSymbologyMask)
End Sub
```

See Also

Storing information
between sessions.....
Reading credit cards
Using the serial port.....
Printing a barcode.....
Receiving power
notifications.....
Enumerating
connections.....
Dialing up a remote
site.....
Calling a BAPI
Posting an IDoc
Queueing soap
requests
Ping.....
Making SOAP calls

SECTION 318

Receiving power notifications

```
Dim Power as ABNOTIFY.Power

Private Sub CreateNotify()
    Set Power =
CreateObjectWithEvents("abNotify.Power", "Power_")
    Power.Enabled = True
End Sub

Private Sub DestroyNotify()
    Set Power = Nothing
End Sub

Public Sub Power_OnPowerOn()
    'do something
End Sub

Public Sub Power_OnPowerOff()
    'do something
End Sub
```

See Also

- Storing information
between sessions.....
- Reading credit cards
- Using the serial port.....
- Printing a barcode.....
- Using the scanner
control
- Enumerating
connections.....
- Dialing up a remote
site.....
- Calling a BAPI
- Posting an IDoc
- Queueing soap
requests
- Ping.....
- Making SOAP calls

Available connections can also be retrieved using a ForEach...Loop as shown below

```
Public Sub ShowConnections()  
  'declare rasenum object  
  Dim rasenum as Object  
  Dim rasname As String  
  
  'create ras enumeration object  
  Set rasenum = CreateObject("abras.rasenum")  
  
  'cycle through enumerations  
  ForEach rasname in rasenum  
    MsgBox rasname  
  Next rasname  
  
  'destroy ras enumeration object  
  Set rasenum = Nothing  
End Sub
```

See Also

- Storing information
between sessions.....
- Reading credit cards
- Using the serial port.....
- Printing a barcode.....
- Using the scanner
control.....
- Receiving power
notifications.....
- Dialing up a remote
site.....
- Calling a BAPI
- Posting an IDoc
- Queueing soap
requests
- Ping.....
- Making SOAP calls

SECTION 319

Enumerating connections

The following programming excerpt demonstrates how to retrieve the names of all available RAS connections configured on the device.

```
Public Sub ShowConnections()  
    'declare rasenum object  
    Dim rasenum as Object  
    Dim i As Integer  
  
    'create ras enumeration object  
    Set rasenum = CreateObject("abras.rasenum")  
  
    'cycle through enumerations  
    For i = 0 To rasenum.Count - 1  
        MsgBox rasenum.Item(i)  
    Next i  
  
    'destroy ras enumeration object  
    Set rasenum = Nothing  
End Sub
```

SECTION 320

Dialing up a remote site

```
Public Sub RasDial()  
    Dim ras as ABRAS.RasConn  
  
    'create ras connection object  
    Set ras = CreateObject("abras.rasconn")  
    'dial connection  
    If Not ras.IsConnected("MyConnection") Then  
        'dial RAS connection  
        ras.Dial "MyConnection"  
    End If  
  
    'hangup connection  
    If ras.IsConnected("MyConnection") Then  
        'hangup RAS connection  
        ras.Hangup "MyConnection"  
    End If  
  
    'destroy connection  
    Set ras = Nothing  
End Sub
```

See Also

Storing information
between sessions.....
Reading credit cards
Using the serial port.....
Printing a barcode.....
Using the scanner
control.....
Receiving power
notifications.....
Enumerating
connections.....
Calling a BAPI
Posting an IDoc
Queueing soap
requests
Ping.....
Making SOAP calls

SECTION 321

Calling a BAPI

```
Private Sub CallBAPI()  
'declare BAPI and Connection objects  
Dim oBapi as ABR3Proxy.BAPI  
Dim oConn as ABR3Proxy.Connection  
Dim oHeadParam, oItemTable As Object  
Dim oReturnParam As Object  
  
'create objects  
Set oBapi = CreateObject("abR3Proxy.BAPI")  
Set oConn = CreateObject("abR3Proxy.Connection")  
  
'set connection properties  
oConn.Destination = "MYSAP"  
oConn.Client = "800"  
oConn.Language = "EN"  
oConn.User = "c0000"  
oConn.Password = "12345"  
  
'set BAPI properties  
oBapi.Name = "BAPI_GOODSMVT_CREATE"  
  
'Add first structure parameter  
Set oHeadParam = oBapi.Params("GOODSMVT_HEADER")  
oHeadParam.Fields("PSTNG_DATE") = "20010101"  
oHeadParam.Fields("DOC_DATE") = "20010101"  
Set oHeadParam = Nothing  
  
'Add second structure parameter  
oBapi.Params("GOODSMVT_CODE").Fields("GM_CODE") = "03"  
  
'Add a simple parameter  
oBapi.Params("TESTRUN") = " "  
  
'create inbound table  
Set oItemTable = oBapi.InTables("GOODSMVT_ITEM")  
  
'Add item fields  
oItemTable.Fields("MATERIAL") = "100-200"  
oItemTable.Fields("STGE_LOC") = "0001"  
oItemTable.Fields("MOVE_TYPE") = "201"  
oItemTable.Fields("ENTRY_QNT") = "5.000"  
oItemTable.Fields("ENTRY_UOM") = "KG"  
oItemTable.Fields("PLANT") = "1001"  
oItemTable.Fields("COSTCENTER") = "0000001000"  
  
'Call BAPI online an commit transaction  
oBapi.Execute oConn, True  
  
'show return parameter from structure
```

```
Set oReturnParam = oBapi.Results("GOODSMVT_HEADRET")
MsgBox "Material Document created " &
oReturnParam.Fields("MAT_DOC")
```

```
'clean up objects
Set oReturnParam = Nothing
Set oItemTable = Nothing
Set oConn = Nothing
Set oBapi = Nothing
```

End Sub

See Also

Storing information
between sessions.....
Reading credit cards
Using the serial port.....
Printing a barcode.....
Using the scanner
control
Receiving power
notifications
Enumerating
connections.....
Dialing up a remote
site.....
Posting an IDoc
Queueing soap
requests
Ping.....
Making SOAP calls

SECTION 322

Posting an IDoc

```
Private Sub PostIdoc()  
    'declare IDOC and Connection objects  
    Dim oIdoc as ABR3Proxy.Idoc  
    Dim oConn as ABR3Proxy.Connection  
    Dim oHeadSegment, oItemSegment As Object  
  
    'create objects  
    Set oIdoc = CreateObject("abR3Proxy.Idoc")  
    Set oConn = CreateObject("abR3Proxy.Connection")  
  
    'set connection properties  
    oConn.Destination = "MYSAP"  
    oConn.Client = "800"  
    oConn.Language = "EN"  
    oConn.User = "c0000"  
    oConn.Password = "12345"  
  
    'set IDOC properties  
    oIdoc.Name = "WMMBID01"  
    oIdoc.Release = "46C"  
    oIdoc.MessageType = "WMMBXY"  
    oIdoc.Sender = "MYSYSTEM"  
  
    'create IDOC segments  
    Set oHeadSegment = oIdoc.AddSegment("E1MBXYH")  
    Set oItemSegment = oIdoc.AddSegment("E1MBXYI")  
  
    'Add header fields  
    oHeadSegment.Fields("TCODE") = "MB1A"  
    oHeadSegment.Fields("BUDAT") = "20010101"  
    oHeadSegment.Fields("BLDAT") = "20010101"  
  
    'Add item fields  
    oItemSegment.Fields("MATNR") = "100-200"  
    oItemSegment.Fields("WERKS") = "1001"  
    oItemSegment.Fields("BWART") = "101"  
    oItemSegment.Fields("LGORT") = "0001"  
    oItemSegment.Fields("ERFMG") = "5.000"  
    oItemSegment.Fields("ERFME") = "KG"  
    oItemSegment.Fields("KOSTL") = "0000001000"  
  
    'transmit idoc online  
    oIdoc.Post oConn  
  
    'show transaction identifier returned by R/3  
    MsgBox "TID returned: " & oIdoc.TransactionId  
  
    'clean up objects  
    Set oHeadSegment = Nothing
```

```
Set oItemSegment = Nothing
Set oConn = Nothing
Set oIdoc = Nothing
```

```
End Sub
```

See Also

- Storing information
between sessions.....
- Reading credit cards
- Using the serial port.....
- Printing a barcode.....
- Using the scanner
control
- Receiving power
notifications
- Enumerating
connections.....
- Dialing up a remote
site.....
- Calling a BAPI
- Queueing soap
requests
- Ping.....
- Making SOAP calls

SECTION 323

Queueing soap requests

```
Dim AsyncPost as ABASYNCPPOST.AsyncPost
Dim SoapProxy as ABSOAP.S SoapProxy

Public Sub InitSoap()
'create soap proxy and async post objects
Set AsyncPost = CreateObject("ABASYNCPPOST.AsyncPost")
Set SoapProxy = CreateObject("ABSOAP.SOAPProxy")
End Sub

Public Sub SavePosts()

'set soap call parameters of the soap proxy
SoapProxy.RouterURL = "http://0.0.0.1:8080/servlet/rpcrouter"
SoapProxy.MethodName = "update"
SoapProxy.ObjectName = "PhoneBook"
SoapProxy.Parameters.Add "name", "Franco Derth", "string"
SoapProxy.Parameters.Add "phone", "555-4321", "string"
'add soap request to asynchronous post queue
'Contacts used as name of application
AsyncPost.Add SoapProxy, "Contacts", "Async ID 1"

'add another soap request to the queue
SoapProxy.Reset
SoapProxy.MethodName = "update"
SoapProxy.ObjectName = "PhoneBook"
SoapProxy.Parameters.Add "name", "Lisa Manns", "string"
SoapProxy.Parameters.Add "phone", "555-4852", "string"
AsyncPost.Add SoapProxy, "Contacts", "Async ID 2"
End Sub

Public Sub RemovePosts()
'count the number of asynchronous posts in the queue
MsgBox "Total posts in queue = " & AsyncPost.Count
Dim item As ABASYNCPPOST.AsyncPostItem
'remove posts added by this application
For Each item In AsyncPost
    If item.AppName = "Contacts" Then
        item.Remove
    End If
Next
End Sub

Public Sub DestroySoap()
'destroy objects
Set SoapProxy = Nothing
Set AsyncPost = Nothing
End Sub
```

See Also

Storing information
between sessions.....
Reading credit cards
Using the serial port.....
Printing a barcode.....
Using the scanner
control.....
Receiving power
notifications.....
Enumerating
connections.....
Dialing up a remote
site.....
Calling a BAPI.....
Posting an IDoc
Ping.....
Making SOAP calls

SECTION 324

Ping

```

Dim oPing As ABSOAPPingLib.SoapPing

Public Sub InitPing()
    'create soap ping object
    Set oPing = CreateObjectWithEvents("ABSoapPing.SoapPing","oPing_")
    oPing.Start
End Sub

Public Sub DestroyPing()
    Set oPing = Nothing
End Sub

Public Sub oPing_OnPingCompletion()
    If oPing.Succeeded Then
        'device is now connected
    End If
End Sub

```

See Also

Storing information
 between sessions.....
 Reading credit cards
 Using the serial port.....
 Printing a barcode.....
 Using the scanner
 control.....
 Receiving power
 notifications.....
 Enumerating
 connections.....
 Dialing up a remote
 site.....
 Calling a BAPI
 Posting an IDoc
 Queueing soap
 requests
 Making SOAP calls

SECTION 325

Making SOAP calls

```

Public Sub RemoteAdd()
'declare soap proxy object using type library
Dim oSoap As ABSOAPLib.SOAPProxy

'create soap proxy object
Set oSoap = CreateObject("ABSoap.SOAPProxy")

'set router url
oSoap.RouterURL = "http://calccompany.com:8080/soap/servlet/rpcrouter"
'set registered method name
oSoap.MethodName = "Add"
'set registered object name
oSoap.ObjectName = "SoapCalc"

'add parameters
oSoap.Parameters.Add "x", "23", "integer"
oSoap.Parameters.Add "y", "46", "integer"

'execute soap request
oSoap.Execute

'check for an error
If (oSoap.SoapFaultExists) Then
    MsgBox ("SoapCalc Error: " + oSoap.FaultString)
End If

MsgBox oSoap.Parameters("x").Value & " + " _
& oSoap.Parameters("y").Value & " = " & oSoap.Result

'destroy soap object
Set oSoap = nothing
End Sub

```

See Also

Storing information
between sessions.....
Reading credit cards
Using the serial port.....
Printing a barcode.....
Using the scanner
control.....
Receiving power

notifications.....
Enumerating
connections.....
Dialing up a remote
site.....
Calling a BAPI
Posting an IDoc
Queueing soap
requests
Ping.....

Copyright © 2001 Abaco.
All rights reserved

The software contains proprietary information of Abaco.; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

Due to continued product development this information may change without notice. The information and intellectual property contained herein is confidential between Abaco. and the client and remains the exclusive property of Abaco. If you find any problems in the documentation, please report them to us in writing. Abaco does not warrant that this document is error-free.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of Abaco.

Microsoft Visual Basic, Microsoft Embedded Visual Basic, Windows®, Windows CE™, Windows 95™, Windows 98™, Windows NT® and Windows 2000™ are trademarks of the Microsoft Corporation.



Abaco Mobile

1100 Northmeadow Parkway
Suite 150, Roswell
Georgia
USA

+1 (678) 319 0105

Internet E-Mail: support@abacomobile.com
"mailto:support@abacomobile.com"

Website: <http://www.abacomobile.com> "http://www.abacomobile.com"

Glossary of Terms

d

data piping

process used by Atoma server to deploy and maintain databes on a client device; desired data is extracted from any number of centralized database servers and then formatted as desired and delivered to a device; after initial creation on device only databases changes (delta) are transferred to the device with each synchronization

S

synchronization parameter

value entered by user before beginning the Tx-Sync process on a client device; value is transferred to the Atoma server and can be read an utilized by a server worker or data filter.

Index

A

abAspex (ASP Extension) • 13
 AbDestroyUI • 597, 609
 abDevio (Scanner, Magnetic Card Reader) • 12, 77
 abFileAttr • 623, 641
 AbFindFirstApp • 603, 607
 AbFindNextApp • 603, 605, 607
 AbGetAppCount • 613
 AbGetAppProperty • 603, 605, 607, 611
 AbGetAppPropertyW • 607, 611
 AbGetSyncInfo • 615
 AbGetSyncLog • 617
 ABHTTPSTATUS • 333, 643
 abImgCap (Image Capture) • 117
 abNotify (Power) • 131
 abR3Proxy (SAP Services) • 139
 abRas (Remote Access Service) • 301
 abShell (Shell Control) • 313
 abSoap (SOAP) • 29, 171, 327
 abSoapPing (Ping) • 403
 abSoapQueue (Web Service Queue) • 29
 abStdio (Serial, Irda, Printer) • 429
 abSys (System Core) • 594
 abUtils (System Tools) • 621
 Add • 63
 Additional Resources • 9
 Agent • 39, 59
 Application

- AppName • 15
- Lock • 17
- AppName • 17
- Unlock • 19
- Lock • 19
- AppName • 19

B

BAPI • 139, 143, 153, 159, 163, 165, 173, 202, 229, 231, 239, 280, 295

ExecuteFunction • 203

InTables • 205

InTableCount • 207

InTables • 207

LoadRequest • 209

Queue • 209

LoadResponse • 211

Queue • 211

Name • 213

ExecuteFunction • 213

Queue • 213

OutTableCount • 215

OutTables • 215

OutTables • 217

ExecuteFunction • 217

LoadResponse • 217

Params • 219

Params • 219

Params • 221

Queue • 223

ResultCount • 225

Results • 225

Results • 227

ExecuteFunction • 227

LoadResponse • 227

Browser

ShowAddressBar • 323

ShowMenuBar • 325

C

Calling a BAPI • 719

Client Workers • 677, 693

Connection

BeginTransaction • 141

Rollback • 141

Commit • 141

Disconnect • 141

Offline • 141

Client • 143

Connect • 143

Commit • 145

BeginTransaction • 145

Offline • 145

Connect • 147

Destination • 147

Connected • 151

Connect • 151

Destination • 153

Connect • 153

Disconnect • 155

Connect • 155

Commit • 155

Rollback • 155

InTransaction • 157

Rollback • 157

Commit • 157

BeginTransaction • 157

Language • 159

Connect • 159

SoapObjectName • 161

URL • 161

Offline • 163

Connect • 163

Password • 165

Connect • 165

Rollback • 167

BeginTransaction • 167

Offline • 167

URL • 169

SoapProxy • 171

User • 173

Connect • 173

Contents

Count • 21

Item • 23

Remove • 25

RemoveAll • 27

ContinueSync • 595, 597, 619

ContinueSyncUI • 597, 609

D

Data Filter Interface • 689

Data Filters • 685, 697

Data Packages • 685, 687

data piping • 735

Delete • 65, 71

DeleteAfterSync • 53

DeleteAll • 67, 73

DeleteHeader • 55

Deploying a client worker • 681

Developing a client worker • 679

devioBARCODE • 85, 93, 647

devioTRIGGERKEY • 95, 645

Dialing up a remote site • 717

E

Encrypted • 47, 61

Enumerating connections • 715

Enumerations • 639

ExecuteSync • 595, 599, 601, 619

ExecuteSyncUI • 601, 609

Exists • 69, 75

F

Field • 228, 233, 271, 287

Name • 229

- Value • 231
- File • 621, 641
 - FindFirst • 623
 - FindFirst • 625
 - FindFirst • 625
 - IsExecuting • 627
 - Load • 629
 - Save • 631
 - StartExec • 633
 - StopExec • 635

G

- GetPayload • 59
- GetPayLoad • 49
- Getting Started • 7

H

- Header • 51
 - Name • 383
 - Value • 385
- Headers
 - Add • 373
 - Clear • 375
 - Count • 377
 - Item • 379
 - Remove • 381

I

- IDoc
 - AddSegment • 175
 - Name • 177
 - IDocType • 177
 - Extension • 179
 - IDocType • 181
 - LoadRequest • 183
 - Queue • 183
 - LoadResponse • 185
 - TransactionID • 185

- Queue • 185
- MessageType • 187
- Post • 189
- TransactionID • 189
- Queue • 191
- LoadResponse • 191
- TransactionID • 191
- Release • 193
- SegmentCount • 195
- AddSegment • 195
- Segments • 197
- AddSegment • 197
- Sender • 199
- TransactionID • 201
- Post • 201

- Introduction • 5

Irda

- CancelRead • 431
- OnDataAvailable • 431
- Config • 433
- DiscoveryTimeout • 433
- InputBufferSize • 433
- PrintMode • 433
- ServiceName • 433
- Connect • 435
- ServiceName • 435
- PrintMode • 435
- DataAvailable • 437
- GetData • 437
- DeviceName • 439
- Disconnect • 441
- Enabled • 441
- DiscoveryTimeout • 443
- Connect • 443
- Enabled • 445

Connect • 445
Disconnect • 445
GetData • 447
Read • 447
InputBufferSize • 449
Read • 449
IsConnected • 451
IsReading • 453
OnConnect • 455
Connect • 455
OnDataAvailable • 457
GetData • 457
OnDisconnect • 459
PrintMode • 461
ServiceName • 461
Connect • 461
Read • 463
OnDataAvailable • 463
CancelRead • 463
ReadMode • 465
Read • 465
ServiceName • 467
PrintMode • 467
Write • 469
Connect • 469
Enabled • 469

J

Java Device Framework • 675
Java Mail Utility • 696, 699

M

Making SOAP calls • 733
MSR

OnDataAvailable • 100
GetData • 100

Separator • 100
DataAvailable • 101
GetData • 101
Enabled • 103
GetData • 105
OnDataAvailable • 107
GetData • 107
Separator • 109
GetData • 109
Supported • 111
WedgeMode • 113
OnDataAvailable • 113
GetData • 113

MSR Notes • 115

N

Name • 41

P

Param

Field • 231
Field • 233
FieldCount • 235
Field • 235
IsSimple • 237
Field • 237
Name • 239
Value • 241

Parameter

Name • 387
Value • 389
Type • 391

Parameters

Add • 393
Clear • 395
Count • 397
Item • 399

- Remove • 401
- Ping • 731
- Pocket PC Device Framework • 11
- Posting an IDoc • 723
- Power
 - Enabled • 133
 - OnPowerOn • 135
 - OnPowerOff • 137
- Printer
 - Advance • 473
 - Font • 473
 - PrintSize • 473
 - StartDoc • 473
 - EndDoc • 473
 - Barcode • 475
 - PrintBarcode • 475
 - BarcodeName • 477
 - Barcode • 477
 - CancelHardwareRead • 481
 - Transport • 481
 - HardwareRead • 481
 - Copies • 485
 - DataAvailable • 487
 - GetHardwareData • 487
 - DrawLine • 489
 - PrintSize • 489
 - EndDoc • 491
 - StartDoc • 491
 - Transport • 491
 - Font • 495
 - PrintText • 495
 - FontName • 497
 - Font • 497
 - GetHardwareData • 501
 - HardwareRead • 501
 - HardwareRead • 503
 - OnDataAvailable • 503
 - CancelHardwareRead • 503
 - ReadMode • 504
 - HumanReadable • 505
 - PrintBarcode • 505
 - PrintSize • 505
 - Indentation • 507
 - PrintText • 507
 - PrintBarcode • 507
 - PrintMaxicode • 507
 - PrintPDF417 • 507
 - StartDoc • 507
 - EndDoc • 507
 - IsReading • 509
 - LabelHeight • 511
 - EndDoc • 511
 - LabelWidth • 513
 - OnDataAvailable • 515
 - HardwareRead • 515
 - GetHardwareData • 515
 - Orientation • 517
 - X • 517
 - Y • 517
 - PrintBarcode • 519
 - Barcode • 519
 - StartDoc • 519
 - EndDoc • 519
 - PrinterType • 523
 - PrintMaxicode • 527
 - Barcode • 527
 - PrintBarcode • 527
 - X • 527
 - Y • 527

StartDoc • 527
EndDoc • 527
PrintPDF417 • 531
Barcode • 531
PrintBarcode • 531
X • 531
Y • 531
StartDoc • 531
EndDoc • 531
PrintRaw • 535
StartDoc • 535
EndDoc • 535
PrintSize • 539
PrintText • 539
PrintBarcode • 539
PrintMaxicode • 539
PrintPDF417 • 539
DrawLine • 539
PrintText • 541
Font • 541
StartDoc • 541
EndDoc • 541
ReadMode • 545
HardwareRead • 545
Spacing • 547
PrintText • 547
PrintBarcode • 547
PrintMaxicode • 547
PrintPDF417 • 547
StartDoc • 549
EndDoc • 549
PrinterType • 549
Transport • 551
EndDoc • 551

HardwareRead • 551
TransportConfig • 553
Transport • 553
PrinterType • 553
X • 555
PrintText • 555
PrintBarcode • 555
PrintMaxicode • 555
PrintPDF417 • 555
DrawLine • 555
Indentation • 555
Y • 557
PrintText • 557
PrintBarcode • 557
PrintMaxicode • 557
PrintPDF417 • 557
DrawLine • 557
PrintSize • 557
Font • 557

Printing a barcode • 709

Q

Queueing soap requests • 727
QueueName • 33

R

RasConn

Dial • 303
IsConnected • 303
HangUp • 305
IsConnected • 307

RasEnum

Count • 309
Item • 311

Reading credit cards • 705
Receiving power notifications • 713
Registry • 635
ReadValue • 637

Request • 37, 63
 Requests • 35, 37
 Requests • 61
 Response • 57
 Responses • 37, 57, 69

Rfc

Execute • 243
 InTables • 245
 InTableCount • 247
 InTables • 247
 LoadRequest • 249
 LoadResponse • 251
 Name • 253
 Execute • 253
 OutTableCount • 255
 OutTables • 255
 OutTables • 257
 Execute • 257
 LoadResponse • 257
 Params • 259
 ParamCount • 261
 Params • 261
 ResultCount • 265
 Results • 265
 Results • 267
 Execute • 267
 LoadResponse • 267
 Queue • 249, 251, 253, 263

S

Samples • 701

Scanner

OnDataAvailable • 77
 GetData • 77
 GetSymbologyMask • 77
 SymbologyMask • 77
 DataAvailable • 79

GetData • 79
 Enabled • 81
 GetData • 83
 GetSymbologyMask • 85
 OnDataAvailable • 87
 GetData • 87
 GetSymbologyMask • 87
 SoftTrigger • 89
 TriggerKey • 89
 Enabled • 90
 Supported • 91
 SymbologyMask • 93
 Enabled • 93
 TriggerKey • 95
 WedgeMode • 97
 OnDataAvailable • 97
 GetData • 97
 SoftTrigger • 645

Scanner Notes • 85, 99

Segment

AddSegment • 269
 Fields • 271
 FieldCount • 273
 Fields • 273
 Name • 275
 SegmentCount • 277
 AddSegment • 277
 Segments • 279

SentRequests • 31

Serial

BaudRate • 561
 Enabled • 561
 CancelRead • 563
 OnDataAvailable • 563
 Config • 565

- BaudRate • 565
- DataBits • 565
- FlowControl • 565
- InputBufferSize • 565
- Parity • 565
- Port • 565
- StopBits • 565
- DataAvailable • 567
- GetData • 567
- DataBits • 569
- Enabled • 569
- Enabled • 571
- BaudRate • 571
- DataBits • 571
- FlowControl • 571
- FlowControl • 573
- Enabled • 573
- GetData • 575
- Read • 575
- InputBufferSize • 577
- Read • 577
- IsReading • 579
- OnDataAvailable • 581
- GetData • 581
- Parity • 583
- Enabled • 583
- Port • 585
- Enabled • 585
- Read • 587
- OnDataAvailable • 587
- CancelRead • 587
- ReadMode • 589
- Read • 589
- StopBits • 591
- Enabled • 591
- Write • 593
- Enabled • 593
- Server Worker Interface • 695
- Server Workers • 693, 697
- Server-Side Development • 683
- SetPayload • 57
- Shell
 - LockAppKey • 315
 - ShowSipButton • 317
 - ShowStartIcon • 319
 - ShowTaskBar • 321
- SigCapture
 - Capture • 119
 - Clear • 121
 - HWindow • 123
 - Image • 125
 - Refresh • 127
 - Image • 127
 - SaveImage • 129
 - Capture • 129
- SoapPing
 - HttpStatus • 171
 - Enabled • 405
 - HttpStatus • 407
 - OnPingCompletion • 407
 - MethodName • 409
 - ObjectName • 411
 - OnPingCompletion • 413
 - Retries • 415
 - Timeout • 415
 - OnPingCompletion • 415
 - RouterURL • 417
 - SoapProxy • 419
 - Start • 421
 - Stop • 423

- Start • 423
- Succeeded • 425
- Retries • 425
- OnPingCompletion • 425
- Timeout • 427
- Retries • 427
- SoapProxy
 - FaultActor • 171
 - FaultCode • 171
 - FaultDetail • 171
 - FaultString • 171
 - ResponseText • 171
 - ResponseXML • 171
 - Result • 171
 - SoapFaultExists • 171
 - ClientCertificate • 329
 - CoerceResults • 331
 - Result • 331
 - Execute • 333
 - RouterURL • 333
 - MethodName • 333
 - ObjectName • 333
 - Result • 333
 - SoapFaultExists • 333
 - FaultActor • 335
 - FaultCode • 337
 - FaultDetail • 339
 - FaultString • 341
 - GetFile • 343
 - RouterURL • 343
 - UseXoomSecurity • 343
 - MethodName • 345
 - ObjectName • 347
 - Execute • 347
- PutFile • 349
- RouterURL • 349
- UseXoomSecurity • 349
- RequestText • 353
- RequestXML • 355
- ResponseText • 357
- Execute • 357
- ResponseXML • 359
- Execute • 359
- Result • 361
- CoerceResults • 361
- RouterURL • 363
- Execute • 363
- SoapFaultExists • 367
- FaultCode • 367
- FaultString • 367
- Execute • 367
- UseXoomSecurity • 371
- RouterURL • 371
- Execute • 372
- GetFile • 372
- PutFile • 372
- Execute • 391
- MethodName • 391
- ObjectName • 411
- RouterURL • 417
- Execute • 643
- PutFile • 643
- SoapQ • 29, 183, 185, 191, 209, 211, 223, 249, 251, 263
- stdioBARCODE • 475, 669
- stdioBAUDRATE • 561, 663
- stdioDATABITS • 569, 661
- stdioFLOWCONTROL • 573, 655
- stdioHARDWARETYPE • 503, 665
- stdioORIENTATION • 517, 667
- stdioPARITY • 583, 659

stdioPORT • 585, 651
stdioPRINTSIZE • 539, 673
stdioREADMODE • 463, 465, 503, 545, 587,
589, 653
stdioSTOPBITS • 591, 657
stdioTEXTFONT • 495, 671
stdioTRANSPORT • 551, 649
Storing information between sessions • 703
Synchronization Guard • 691
synchronization parameter • 735
Synchronization Parameters • 689, 691, 696, 697

T

Table

MoveFirst • 280
MoveNext • 280
MoveLast • 280
Fields • 280
AddRecord • 281
Fields • 281
EOF • 283
FieldCount • 285
Fields • 285
Fields • 287
MoveFirst • 289
MoveNext • 289
MoveLast • 291
MoveNext • 293
Fields • 293
Name • 295
RecordNumber • 297
RecordCount • 299
AddRecord • 299
TxSyncCallbackFunc • 595, 599, 619

U

URL • 43
UserDef • 45, 59
Using the scanner control • 711
Using the serial port • 707

Contents

| | |
|-----------------|----------|
| Overview | 3 |
|-----------------|----------|

| | |
|----------------------------|----------|
| System Installation | 5 |
|----------------------------|----------|

| | |
|----------------------------|----|
| Server Requirements | 8 |
| Installed Components | 12 |

| | |
|------------------------|-----------|
| Getting Started | 15 |
|------------------------|-----------|

| | |
|----------------|-----------|
| Console | 17 |
|----------------|-----------|

| | |
|------------------------------------|----|
| Connection | 19 |
| Adding a Connection | 21 |
| Resetting a Device Sync | 23 |
| Removing a Connection | 25 |
| Testing a Connection | 27 |
| Connection Info | 29 |
| IDocs | 31 |
| Downloading an IDoc | 33 |
| Removing an IDoc | 35 |
| IDoc Filters | 37 |
| Server | 41 |
| Error Log | 43 |
| Statistics | 45 |
| Soap Proxies | 47 |
| Soap Proxy | 49 |
| Message Log | 51 |
| Document Server | 53 |
| Starting the Document Server | 57 |
| Stopping the Document Server | 59 |
| Message Log | 61 |
| Storing IDoc to a Database | 63 |

| | |
|--------------------------|-----------|
| Glossary of Terms | 65 |
|--------------------------|-----------|

| | |
|--------------|-----------|
| Index | 67 |
|--------------|-----------|

CHAPTER 1

Overview

Introduction

The Atoma R/3 Connector is an open standards-based tool that exposes all the functionality of R/3 as a Web Service to any platform or system capable of using standard web technology without the need for ABAP programming knowledge. The R/3 Connector features web-based configuration of R/3 system, functions and documents (RFC, BAPI, IDOC) as well as connection management and pooling, integrated transaction and security model, server-based java classes, and platform and language independence.

The R/3 Connector also includes a document server which can establish itself as an ALE partner with R/3, making it possible to receive documents (IDOCs) asynchronously and map them to tables in a database of your choice or simply to XML files. These documents (IDOCs) can contain large amounts of information, for example material master and updates, that can be automatically populate a database making this information available throughout your enterprise.

The main objectives of this document are to help you understand and configure the R/3 Connector, as well as show some examples of how to use it from a Java application. You should be familiar with following terms and technologies: SAP R/3, RFC, IDOC, BAPI, Java, XML, SOAP.

CHAPTER 2

System Installation

In This Chapter

| | |
|----------------------------|----|
| Server Requirements | 8 |
| Installed Components | 12 |

Server Requirements

The minimum server requirements are as follows:

Hardware System

- Network Interface Card
- 140 MB Free Hard Disk Space
- 128 MB RAM

Additional RAM, Hard Disk Space and Processor Clock Speed may be required as the number of mobile clients serviced by a given server is increased.

Operating Systems

- Windows 2000 / XP
- Sun Solaris Sparc 2.8
- HP-UX 11x
- IBM AIX Power

Databases

In order to store IDocs using the R/3 Connector Document Server a server side a DBMS may be used. The following databases are supported:

- Oracle 8i
- SQL Server 7.0, 2000
- DB2 7.1, 7.2

Also one of the following drivers may be used to connect to the database and load the initial data during the installation:

- JDBC - Include the desired driver files into the server's CLASSPATH environment variable. This is the **recommended** type of driver for Atoma.
- ODBC - Create a valid DSN for the created Atoma database. Even if the Document Server runs with this type of driver it is **not recommended**.

SAP Java Connector (JCO)

The R/3 Connector uses the SAP Java Connector (JCO) to execute function calls and receive IDocs. Before you start the installation install JCO on the target server. Make sure that the jCO.jar file is included in the server's CLASSPATH environment variable.

Required Installation Permissions

If the target platform is Win32 the installation will create a new service to start the R/3 Connector. Make sure that the Account used for the installation has permissions to create Win32 services.

R3/Connector Listening Port

The user will be prompted for the R3/Connector listening ports during the installation process. By default the ports used are 9080 and 9087. Make sure that this ports are available, if not then change them for two available ports.

Authentication Providers (LDAP)

The R3/Connector avoids the administrative nightmare of creating and managing multiple user databases by seamlessly integrating with user management systems already in use by an enterprise. Any of the following providers are supported:

- Microsoft Active Directory.
- Netscape Directory Service.
- SAP Directory Service
- IBM Secure Way.
- Most other LDAP compliant providers.

A group that contains the users with administrative permissions to the R3/Connector needs to be created in the selected Directory Service system. If you are using the SAP Directory service you need to create a new object and assign this object to the desired R3/Connector administrators.

Installed Components

The following third party products will be included as part of the installation.

Java Development Kit (JDK)

Following is a list of the JDK's installed per platform:

- Win32 - Sun JKD 1.3.1 for Microsoft Windows.
- Sun Solaris - Sun JDK 1.2.2 for Solaris
- HP-UX - HP JDK 1.2.2 for HP-UX
- IBM AIX Power - IBM JDK 1.2.2 for AIX Power

Support for other platforms is planned for future releases.

Tomcat 4.0

The Atoma R/3 Connector runs on top of the Tomcat 4.0 Servlet Container.

Supporting Libraries

The following third party libraries are installed as part of the Atoma framework:

- Xerces 1.4.1 - Xml Parser that is composed of the following files: xerces.jar.
- Apache Soap 2.2 - RPC Router that is composed of the following file soap.jar.
- Java Cryptography Extension 1.2.1 - Composed of the following files: jce1_2_1.jar, local_policy.jar, sunjce_provider.jar, US_export_policy.jar.
- Java Secure Socket Extensions 1.0.2 - Composed of the following files: jcert.jar, jnet.jar, jsse.jar.
- Java Mail 1.2 - Composed of the following files: imap.jar, mail.jar, mailapi.jar, pop3.jar, smtp.jar.
- Java Activation Framework 1.0.1 - Composed of the following file: activation.jar.
- Ldap 1.2.3 (Only for Platforms running JDK 1.2.2) - Composed of the following files: jass.jar, ldap.jar, ldapbd.jar, providerutil.jar.
- JNDI 1.2.1 (Only for Platforms running JDK 1.2.2) - Composed of the following file: jndi.jar.

If a different version of any of these libraries is already in the target server's CLASSPATH environment variable please remove them. These may cause the R/3 Connector not to work properly.

CHAPTER 3

Getting Started

Win32 Platforms

Once the Atoma R/3 **Connector** installation has been completed, a service called R3Connector_XXXX (where XXXX is the port number selected during installation) will be created. This service is set to the automatic startup mode. This means that once you reboot your computer the R/3 **Connector** server will start running. You can also start the R/3 **Connector** server manually by going to the windows Service Control Manager and start the R3Connector_XXXX service. To access the R/3 **Connector** console open the Console program from the R3 Connector programs menu or alternatively start you web browser and enter the following address: `http://localhost:XXXX/r3connector/index.jsp` where XXXX is the port number selected during installation. If you are accessing the console from a different computer replace localhost by the R/3 **Connector** server host name.

Unix Platforms (Solaris, HP-UX, etc...)

Before you start the Atoma server make sure the following environment variables are loaded: R3CONNECTOR_HOME and R3CONNECTOR_CATALINA_HOME. The value of R3CONNECTOR_HOME is the R/3 **Connector** installation directory "/r3connector", the value of R3CONNECTOR_CATALINA_HOME is the Tomcat installation directory "/r3connector/tomcat". To start the **Connector** server you have to start the Tomcat web container. To do this go to the /r3connector/tomcat/bin folder and execute the startup.sh file. To stop the **Connector** server execute the shutdown.sh file in the same folder. To access the **Connector** console start your web browser and enter the following address: `http://localhost:XXXX/r3connector/index.jsp` where XXXX is the port number selected during installation. If you are accessing the console from a different computer then replace localhost by the R/3 **Connector** server host name. To uninstall R/3 **Connector** execute the following file: `/r3connector/_uninst/uninstall.sh`.

To get started using the **Connector**, simply follow the instructions on how to add a new connection to an SAP R/3 system. Once a connection has been configured and tested you are ready to start calling RFCs , BAPIs and posting IDocs to that same system. The Atoma device framework includes a control called the **abR3Proxy** that simplifies the development of programs that execute RFCs and BAPI's and post IDocs from the client device to R/3, via the R/3 Connector SOAP Proxy.

Prerequisites

- Understanding of SAP R/3, namely RFCs, BAPIs, IDocs, and ALE.
- Understanding of Java, its technologies, and some programming experience.
- Understanding of Web-based technologies and terms.

CHAPTER 4

Console

The Atoma R/3 Connector **Console** is a browser based application that serves as the central administration interface for the Connector. Virtually all of the system's configuration options are accessible through the Console. Most changes made through the Console are instantly activated and do not require a stop and restart of the R/3 Connector server.

Console Login and Logout

A Console user must be authenticated before being allowed to access the application. When the Console is first opened a Login screen is displayed where valid login information must be entered before the Console application will be opened. The information entered will be validated against the Ldap group designated as system administrators during installation of the system (see System Installation).

Once a user is validated, a Console session is created and the user may start configuring the system. The session will timeout and be terminated after a period of inactivity. Once the session is terminated, the user will be required to login and create a new session before accessing the Console again. The default timeout time is 30 minutes. This can be changed in the Tomcat Configuration, for more information refer to the Tomcat documentation: http://localhost_XXXX/tomcat-docs.

A **Logout** option is provided in the top right side of the console. When this option is selected the session is terminated. A user should logout before leaving the Console unattended to prevent unauthorized access.

Console Menu

The **Menu** is displayed at the top of the browser's window after the Console is opened. Each item listed in the Menu is a hyper-link to a distinct R/3 Connector configuration option. The main menu options are:

- **Connections** - Contains the configuration options specific to an R/3 system. With one R/3 Connector server you can manage connections to several R/3 systems.
- **Server** - Contains options global to the R/3 Connector server.
- **Soap Proxy** - Contains the configuration options for the R/3 Connector's Soap Proxy.
- **Document Server** - Contains the configuration options for the R/3 Connector's Document Server.

Using the Console

After entering information and/or changing parameters through the Console, your changes must be activated. To activate changes an **Apply** button/hot-spot is provided on each module where changes can be made. After the **Apply** button is clicked, your changes either become effective immediately or are stored and will be active when the specific component is restarted.

To remove items from your server settings and configurations a removal icon is usually provided. When the removal icon is clicked, you are prompted to confirm the removal with a message box. If the removal is confirmed through the message box, the item is deleted and is effectively removed from the system.

Navigation between the options of the Console is achieved through the links that are provided on each screen displayed. A **Back** button/hot-spot is provided on many screens to allow you to return to the previous screen. While the **Back** button of your browser may also allow you to return to the previous screen in most cases, it is usually best to use the hot-spot provided on the screen.

In This Chapter

| | |
|-----------------------|----|
| Connection | 19 |
| IDocs..... | 31 |
| Server | 41 |
| Soap Proxies..... | 47 |
| Document Server | 53 |

Connection

The **Connection** menu option provides access to Atoma R/3 Connector's connection management features. In order to access an SAP R/3 system, download function or document metadata, a proper R/3 connection needs to be configured.

Connection Info

| | |
|--------------------|---|
| User | Valid user name to log on against R/3 such as "c1234567" |
| Password | Valid password to log on against R/3 such as "secret". |
| Client | R/3 Client number, such as "000" or "003" |
| Language | Valid SAP Language string, such as "EN" or "DE". |
| Trace | Controls low-level RFC tracing. A value "0" turns off tracing, while value "1" turns on tracing. The default is "0" or off. |
| Use Load Balancing | Select this option if you are connecting to an SAP System using Load Balancing |

SAP Host Info

| | |
|--------|---|
| ASHOST | The connection string for the SAP R/3 Application Server host. You can specify the TCP host name, such as "r3.company.net" or a valid ip address. You can include a router in the path using "/H/" as a separator such as "/H/204.79.199.5/H/194.45.237.230". |
| GWHOST | The connection string for the SAP R/3 Gateway host. You can specify the TCP host name, such as "gw.company.net" or a valid ip address. You can include routers in the path using "/H/" as a separator. |
| GPSERV | SAP R/3 system number, such as "00". |

| | |
|----------------------|--|
| System Number | SAP R/3 system number, such as "00". |
| SAP Host Info | |
| MSHOST | Host Name of the message server. |
| R3 Name | Name of the R3 System. |
| Group | Name of the group of application servers. |
| Page Options | |
| Apply | Makes configuration changes permanent. |
| Test | Attempts to create a connection with the configured parameters. |
| IDocs | Displays the IDocs configuration option for the current configuration. |
| Remove | Removes the current connection configuration from the Connector . |

CHAPTER 5

Adding a Connection

The **Connector** allows you to maintain multiple SAP R/3 systems which can simultaneously be used to execute functions and post idocs. This flexibility now only permits quick access to multiple R/3 systems, it can also be used to update and maintain information enterprise wide.

To add a new Connection

- 1** To add a new connection select the Connections->New menu option.
- 2** Enter a unique, well-formatted name in the text box provided and Click Ok.
- 3** Fill out the connection and host information as described in the Connection page.
- 4** Click on the **Apply** button to make the changes permanent.

To make sure that the connection information is accurate click on the **Test** button.

Resetting a Device Sync

To edit an existing Connection

- 1 Select the Connections->(desired Connection) from the main menu.
- 2 Edit the connection information and click the **Apply** button.

Removing a Connection

The **Removal** option allows you to delete connection registrations from an installation. When a connection is deleted all references to the connection on the installation are removed and the connection may no longer perform function calls or posting documents to the server.

To Remove a Connection

- 1** Select the Connections->(desired Connection) from the main menu.
- 2** Click on the Removal icon on the Connection screen.
A message box will appear prompting you to confirm the deletion.
- 3** Click the OK button to confirm to delete the connection.
If you do not wish to delete the connection click the Cancel button.

Testing a Connection

To test an existing Connection

- 1** Select the Connections->(desired Connection) from the main menu.
- 2** Click on the **Test** button in the Connection page.

Connection Info

The connection parameters provided in this section will only be used to download document (IDOC) metadata, and the document server (ALE). Every request that is processed by the Atoma R/3 Connector through SOAP will contain its own connection string, specifying username, password, client, language and destination to ensure security and authentication for every transaction.

IDocs

The **IDocs** page provides access to the **Connector's** powerful document mapping services. Functioning as an ALE Server, the **Document Server** can receive an outbound IDoc from R/3 and map its segments and fields to tables and fields in a database or simply to an xml file. This simple, yet flexible process provides immediate access to document information, like material master data and its updates, through simple database queries. If you combine the **Document Server** to Atoma's Data Piping services, then you have a powerful combination of services that will update any information on the client device databases as soon as it is available from R/3. Also this page provides the option to create a filter for each configured IDoc. The advantages of creating a filter are the following:

- 1 Unnecessary segments and fields are not stored in the database or added to the XML file. This minimises the processing time of each received IDoc, saves database space and makes the XML file easier to read.
- 2 Provides the option to use Alias names for each segment and field. This provides the advantage of creating the database tables with names that are significant to the system being implemented instead of using names as E1MARAM and MATNR. Also this Aliases are used when creating the XML file making the file easier to understand.

IDoc List

| | |
|------------|--|
| IDoc Type | The IDoc type indicates the SAP format that is to be used to transfer the data for a business transaction. |
| Basic Type | Some IDoc types are supplied by SAP in the standard system: these are referred to as basic types. |
| Extension | Other IDoc types are customer extensions: in these cases, a basic type is combined with an extension which is created by the customer, according to certain rules. |
| Release | Version of the R/3 IDoc that will be used. An R/3 system can understand different versions of an IDoc, for example the system can be 46D and receive 46C IDocs. |

Page Options

| | |
|----------|---|
| Remove | Removes the selected IDoc metadata from the Document Server. IDocs can be selected by checking the Remove box besides each one. |
| Download | Downloads an Idoc metadata and makes it available for posting and mapping to a database through the Document Server or to an xml file. |
| Back | Goes back to the previous page. |

Downloading an IDoc

Downloading an **IDoc** means that the **Connector** will then try to connect to R/3 using the configured connection parameters and download the segments, fields and structures and make up the **IDoc**.

To Download an IDoc

- 1** Select the Connections->(desired Connection) from the main menu.
- 2** Click on the IDocs link.
- 3** Enter the IDoc Type, Extension Type, Basic Type and Release text boxes provided.
- 4** Click on the **Download** button and wait for a confirmation message.

Removing an IDoc

The **Removal** option allows you to delete IDocs from an installation. When an IDoc is deleted all references to that IDoc are removed from the system including the IDocs filter information.

To Remove an Idoc

- 1** Select the Connections->(desired Connection) from the main menu.
- 2** Click on the IDocs link.
- 3** Click on the **Remove** check box located in the left-hand side of the IDoc to be removed.
- 4** Click on the **Removal** icon at the bottom of the IDoc list.
A message box will appear prompting you to confirm the deletion.
- 5** Click the **OK** button to confirm to remove the IDoc.
If you do not wish to delete the IDoc click the **Cancel** button.

IDoc Filters

Creating an IDoc Filter provides the option to avoid unnecessary IDoc segments and fields from being inserted to a database table or been saved to an XML file. Also provides the option of creating Alias names for each segment and field. If no filter is created for an IDoc then the database tables or XML file structure will be created using all the IDoc's segments and fields and it's default names.

To Create an IDoc Filter

- 1 Select the Connections->(desired Connection) from the main menu.
- 2 Go to the IDocs page.
- 3 Click on the desired IDoc.
- 4 Select only the Segments and Fields you want to be inserted into the database or saved to the XML file (By default all the segments and fields are selected). More information on how to do this is below.
- 5 Assign the Alias Names for your desired segments and fields. More information on how to do this is below.
- 6 Click **Apply Filter** option at the top of the screen to make the changes permanent.

To Create the Filter Tables

- 1 In the IDoc Filter screen click **Create DB Tables** option at the top of the screen to create the database tables in the database configured in the **Document Server** screen. A message box will appear prompting you to confirm the tables creation.
 - 2 Click the **OK** button to create the tables. If you do not wish to create the tables click the **Cancel** button.
- * When the database tables are created any existing table with the same name as any of the segments in the filter will be dropped. A table will be created for each segment and its name will be the segment Alias Name. More information on the table structure can be found in the **Document Server** help page.

To Include/Exclude a Field in the Filter

- 1 In the IDoc Filter screen go the desired segment.
- 2 Expand the segment to display all it's fields.
- 3 Click the check box at the left hand side of the Field name to include/exclude the Field in the filter.
- 4 To include/exclude all the Fields in a Segment click the All check box at the right hand side of the segment name.

To Include/Exclude a Segment in the Filter

- 1 In the IDoc Filter screen go to the desired segment and make sure that at least one of the fields is selected. If there are no fields selected for a segment then a table for that segment will not be created and that segment will also be excluded from the XML file.

* There is one scenario where the segment may not contain any selected fields but its table will be created. This happens if this segment contains a child segment that has fields selected. Because the table for this child segment will be created, then the table for its header segment will also be created containing only the control fields. This is done to provide the user a way to reconstruct the IDoc hierarchy. More information about IDoc hierarchy and table control fields may be found in the Document Server's help page.

To Change the Alias Name of a Segment

- 1 In the IDoc Filter screen go to the desired segment.
- 2 Enter the desired segment Alias Name in the Alias text box at the right hand side of the segment name. The Alias name by default is the segment name.

To Change the Alias Name of a Field

- 1 In the IDoc Filter screen go to the desired segment.
- 2 Expand the segment to display all its fields.
- 3 Enter the desired Field Alias Name in the Alias text box at the right hand side of the field name. The Alias name by default is the field name.

To Change the Alias Name of a Field

- 1 In the IDoc Filter screen go to the desired segment.
- 2 Expand the segment to display all its fields.
- 3 Enter the desired Field Alias Name in the Alias text box at the right hand side of the field name. The Alias name by default is the field name.

Changing the Nullable Property of a Field

- 1 In the IDoc Filter screen go to the desired segment.
- 2 Expand the segment to display all its fields.
- 3 Check/Uncheck the nullable property of the desired field. If the nullable property of the field is not checked then the field will be created as Not Null in the database. Make sure that R/3 always populates this field or the document server will receive a db error when inserting the data.

Other Page Options

| | |
|-------------------|---|
| Expand All | Expands all the segments in the Filter, displaying all the fields for each segment. |
| Close All | Closes all the segments in the Filter, hiding all the fields for each segment. |
| Back | Goes back to the previous page. |
| Select All Fields | Selects all the fields in the IDoc to be included in the filter. |
| Clear All Fields | Removes all the fields of the IDoc from the Filter. |

CHAPTER 6

Server

The Server menu option contains the R/3 Connector general options. These options are:

- 1 Error Log - This module provides the user with a general point to check for any error from any component in the R/3 Connector server.
- 2 Statistics - Displays some server utilization information.

In This Chapter

| | |
|------------------|----|
| Error Log | 43 |
| Statistics | 45 |

Error Log

The Connector Error Log provides a mechanism to the end user to see in a central place the error messages that any of the Connector pieces may have. This option may be of great help when problems are encountered with the **Connector** and help from the vendor is required.

Checking for Error Messages

- 1** Go to the Tools page.
- 2** Click on the **Error Log** option. A list of all the existing error log files is displayed. A Error Log file is started each time the connector is started.
- 3** Find the desired Error Log file (usually the newest one) and click on it.
- 4** A list of all the error logged for that session is displayed.

Statistics

The **Statistics** page displays information about the current state of the Connection in order to better understand what it is doing, how many connections to the database it has at the moment. This information is purely to help implementators optimize their system.

Memory Usage

| | |
|--------------|--|
| Total Memory | Total amount of memory available to the Java Virtual Machine. |
| Used Memory | Amount of memory in use by the Java Virtual Machine. |
| Free Memory | Amount of memory that is either not in use or has been released by the Java Virtual Machine. |

Connection Pool

| | |
|----------------------|--|
| Inactive Connections | The number of database connections in the pool that are not currently in use. |
| Active Connections | The number of database connections in the pool a that are currently being used to process IDocs. |
| Total Connections | Total amount of database connections in the pool. |

Page Options

| | |
|----------|--|
| Refresh | Queries the system for the most current performance information. |
| Force GC | Suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. The Virtual Machine automatically recycles unused objects and using this option is not necessary for the operation of the Connector. |

CHAPTER 7

Soap Proxies

The Soap Proxies menu option contains the configuration pages for SOAP connections from the client to R/3.

In This Chapter

| | |
|-------------------|----|
| Soap Proxy | 49 |
| Message Log | 51 |

Soap Proxy

The Connector Soap Proxy makes it possible to expose R/3 as a secure Web Service because it is based on HTTP(s) and XML. Any platform or device that is capable of having an Internet browser can post transactions to R/3 using our Connector. No ABAP programming is required, only knowledge of the functions or documents that are to be used and what data is needed to execute the transaction. This proxy goes hand by hand with the abR3ConnectorProxy client deployed with Atoma to any supported device. This library provides an easy way to execute RFC's and BAPI's also to post IDoc's to any SAP system. The documentation on how to use this abR3ConnectorProxy client can be found in the Atoma Client SDK documentation.

Soap Proxy

| | |
|---------------------------------------|---|
| Close Idle Connections After | The number of minutes the Connector will wait to close a connection without any recent activity. This means that if a client opens a connection to SAP and leaves the connection idle for more than this time, that connection will be closed. |
| Check For Idle Connections | The number of minutes the Connector will wait to check if there are connections that have not had any recent activity, and thus, need to be considered idle. |
| Maximum Time to wait for a Connection | The maximum time that the Proxy will wait for a connection from SAP. If a connection request is received by the Proxy it will ask SAP for a connection and will wait a maximum of this time until it will timeout. |
| Maximum Connections Pool Size | The maximum number of connections to R/3 that can be created in the Connector pool. This is the maximum number of simultaneous connections to SAP by the Soap Proxy. |
| Trace Level | Controls low-level RFC tracing. A value "0" turns off tracing, while value "1" turns on tracing. The default is "0" or off |

Debug Mode

Determines whether debugging messages will be sent to standard output during normal operation of the **Connector**. The default is *OFF*. Note that enabling this option will incur in some overhead to performance. This option will take effect immediately, it does not require to apply the changes.

JCO Trace

Determines whether the JCO trace messages will be active or not. The default is *OFF*. Note that enabling this option will incur in some overhead to performance. This option will take effect immediately, it does not require to apply the changes.

Page Options

Apply

Makes configuration changes permanent.

Message Log

Displays SOAP proxy request messages generated when the **Connector** is running.

Message Log

The **Message Log** option displays status and error messages that have been output by **Soap Proxy** during normal operation. For each message the following is displayed: (i) **Event Time** - data and time that message was generated; **Message** - status information or error description.

Document Server

The **Connector** includes a **Document Server**, which is a service that acts as an ApplicationLink Enabling (ALE) server responsible for receiving documents, namely outbound IDocs from R/3, and mapping them to tables in a database or storing them to an XML file. This service makes it possible to receive new material master updates while your system is running and makes them available immediately with a simple database query. To learn more about how the IDocs are mapped see the IDocs section.

If the **Document Server** is configured to store the received IDoc's to a database it will connect to the selected database and store the IDoc information to the tables defined in the IDoc filter on the IDocs page. If the **Document Server** cannot connect to the database or there is an error while inserting the IDoc information it will store the IDoc to disk. A process called **Failed IDocs Listener** that is started with the **Document Server** will run at a given interval of time and retry to insert this failed IDocs to the database. These temporarily failed IDoc are stored to the following folder */r3connector/documentserver*. Do not remove any Failed IDoc from this folder unless you are sure that it cannot be inserted to the database, usually the **Failed IDocs Listener** will process them. The process described above provides a mechanism to handle database connection interruptions without any loss of data.

If the **Document Server** is configured to store the received IDoc's to an XML file it will create a new XML file for every received IDoc and save it to the configured destination folder. The IDoc information will be saved to the XML file following the filter created in the IDocs configuration page.

Note that only one **Document Server** per **Connector** installation is currently supported. Therefore, the **Document Server** page displayed in all the connections refers to the same server configuration. It is just provided for quick access. Also, In order for the **Document Server** to work properly a *Partner Profile* and *Logical System* must be configured on the SAP R/3 side.

Document Server

SAP Configuration

This list displays all the available connection configurations in the **Connector**. The **Document Server** will use the chosen connection in this list to access R/3.

| | |
|------------------------------|---|
| Program ID | The Program Identifier is the name that the document server will use to identify itself to the R/3 system when listening for documents. This name is the same as the SAP R/3 <i>Logical System</i> which has to be configured for the document server to work properly. |
| Listeners | Number of concurrent threads or workers that will be listening for new documents (IDocs) from R/3. Default and minimum value is "1". |
| Check for failed IDocs every | The amount of time in <i>minutes</i> that the Document Server checks for documents that have been downloaded but have not yet been populated in the configured database. |
| Start Document Server | Determines whether the Document Server will be started automatically when the Connector service is started or not. |
| Store Received IDocs to | Determines the where the received outbound IDoc's from R/3 will be stored. If the Database option is selected then the IDoc's will be stored to the configured database, otherwise if the Xml File option is selected then each received IDoc will create an XML document containing the IDoc information in the configured destination folder. |

Database Info

| | |
|-------------------|--|
| Database Driver | JDBC database driver name, such as "com.merant.datadirect.jdbc.sqlserver.SQLServerDriver". |
| Database URL | A JDBC database URL of the form jdbc:subprotocol:subname, such as "jdbc:merant:sqlserver://localhost:1433" |
| Database User | A valid user for the configured database. |
| Database Password | A valid password for the database user. |

Folder Info

| | |
|------------------|---|
| Destination Path | The folder where the Document Server will store the received IDoc's xml files.. |
|------------------|---|

Page Options

| | |
|-------------|---|
| Start | Immediately starts the Document Server if it is not already started. |
| Stop | Immediately stops the Document Server if it is currently running. |
| Message Log | Displays status and error messages generated when the Document Server is running. |
| Test DB | Attempts to connect to the configured database to ensure proper operation of the Document Server . |
| Apply | Makes configuration changes permanent. |
| Back | Goes back to the previous page. |

Starting the Document Server

To start the **Document Server**, select the DocumentServer menu option. Then click on the **Start** hotspot which is on the same line as the where *Start Document Server* options are listed.

If the Document Server does not start, look at the **Message Log** (just a couple of lines above the Start hotspot) to see any error messages that were generated.

Stopping the Document Server

To stop the **Document Server**, simply select the **Document Server** menu option. Then click on the **Stop** hotspot which is on the same line as the where *Start Document Server* options are listed.

If the Document Server does not stop properly, look at the **Message Log** (just a couple of lines above the Start hotspot) to see any error messages that were generated.

Message Log

The **Message Log** option displays status and error messages that have been output by **Document Server** during normal operation. For each message the following is displayed: (i) **Event Time** - date and time that message was generated; **Message** - status information or error description.

Storing IDoc to a Database

To store the IDoc information to a database the Connector needs to create a series of tables.

IDoc Control Table

A control table is created that is global to all the configured IDocs. This control table is called IDOC_CONTROL and contains the following fields:

| | |
|----------------|---|
| TID | IDoc's Transactional ID |
| IDOC_TIMESTAMP | The date and time when the IDoc was received |
| DIRECTION | 1 - outbound IDoc |
| IDOC_STATUS | Reserved for future versions |
| IDOC_NUMBER | SAP's IDoc Number |
| MESSAGE_TYPE | The received IDoc message type |
| SENDER_NAME | The name of the logical system that sent the IDoc |
| SENDER_TYPE | The IDocs sender type |

This table is used as a log of all the stored IDocs in the database. Every time you select the Create Tables option from the Create Filter page the system will check if the IDOC_CONTROL table exists, if it does not exist then the Connector will create it. The data on this table is for historical purposes only and can be purged when necessary.

Segment Tables

A table is created for every selected segment in the IDoc filter. This name of this table will be the Alias name given to the segment. It will contain only the fields selected in the filter for the segment and the name of each field will be its alias. Additional to the selected fields the table will contain a couple of control fields which are the following:

| | |
|-------------|---|
| IDOC_NUMBER | SAP's received IDoc number. This is used as the unique identifier across all the segments of the received IDoc. |
| SEGMENT_ID | Unique to each of the segments of the received IDoc. This number is reset to 1 for each received IDoc. |
| PSEGMENT_ID | Contains the segment id of the parent of the segment. For example if this field contains "1" it means that the table with SEGMENT_ID "1" is its parent. The number "0" as the PSEGMENT_ID means that it does not have any parent. |

With the information stored in this 3 control fields the developer can recreate the IDoc hierarchy.

An index is created for each of the segment tables. This index contains the 3 control fields. Depending on the implementation more indexes can be added to the table by the database administrator. Also all the segment SAP fields are created as NULLABLE by default. If for some reason any of these fields needs to be changed to NOT NULL it can be done without any problems for the **Document Server** (just make sure that null values will not be sent by SAP on the desired field).

Glossary of Terms

B

BAPI

business application programming interfaces that provide a standard for business interfaces though an object-oriented view of R/3 application modules.

C

classpath

Java® virtual machine variable that specifies the location of Java® classes such that they may be loaded by the virtual machine when referenced

I

IDOC

containers of data with a hierarchical structure composed of segments and fields.

J

JDBC

Java Database Connectivity; an API for accessing relational and tabular data (primarily databases) from the Java® programming language

JDBC URL

string that specifies the location of a JDBC data source

R

RFC

remote function calls are procedures located in an SAP R/3 system that can be executed between two SAP systems or an SAP and an external system.

S

SOAP

Simple Object Access Protocol; lightweight mechanism of information exchange between remote applications

SOAP router

server application that listens for incoming SOAP messages; SOAP router is responsible for executing or forwarding SOAP messages as needed and returning required responses to the requestor.

Index

A

Adding a Connection • 15, 21

B

BAPI • 15, 65

C

classpath • 65

Connection • 19, 27, 33

Connection Info • 19, 29

Console • 17

D

Document Server • 53

Downloading an IDoc • 33

E

Error Log • 43

G

Getting Started • 15

I

IDOC • 15, 53, 65

IDoc Filters • 37

IDocs • 31, 35, 37, 53

Installed Components • 12

J

JDBC • 65

JDBC URL • 65

M

Message Log • 51, 61

O

Overview • 3

R

Removing a Connection • 25

Removing an IDoc • 35

Resetting a Device Sync • 23

RFC • 15, 65

S

Server • 41

Server Requirements • 8

SOAP • 65

Soap Proxies • 47

Soap Proxy • 49

SOAP router • 65

Starting the Document Server • 57

Statistics • 45

Stopping the Document Server • 59

Storing IDoc to a Database • 63

System Installation • 5, 17

T

Testing a Connection • 27

Data Piping

Data Piping Process

Description

The Data Piping Process extracts data from Enterprise Data Centers and creates packages of information to be sent to the devices in order to synchronize its database. In addition, maintains a local image of the device database in order to send just only changes occurred since the last synchronization took place.

Uses Data Package, Database Source and Filtering Component definitions created using the Data Piping Configuration.

It is synchronized process invoked for a specific device. The invocation also set the way the exceptions are handled during processing.

The process has the following steps:

- Extracts parameters from the Invocation Message.
- Create a list of Data Packages to be synchronized in accordance to the group identifier received as parameter.
- Check if the device belongs to the same group when it was processed the last time before and if the filtering data was the same. If the device has changed of group or filtering data, the device data must be recreated.
- The Packager is initiated.
- Create a list of Data Packages that must be purging from the Local Database due changes into the list of Applications in use by the group or changes of the list of Data Packages in use by the Applications.
- For each item of the previous list, purges data on the Local Database and create and add the correspondent XML document to the package that will be sent to the device. It implies to drop a table for each Data Package and delete the record with the references to this table.
- For each item of the Data Packages to be synchronized process the data from the Enterprise Data Center and create and add the correspondent XML document to the same package to be sent to the device. It includes:
 1. Invokes via SOAP the Filtering Component to get the filtering condition to be added at the WHERE condition of statement used to inquire the Data Center.
 2. Using the Datasource attribute of the Data Package, create a Datasource definition and use that to open a connection to the Data Center.
 3. Create a connection to the Local Database where resides an image of the data already has the device.
 4. Execute the query to get the data to be processed from the Data Center.
 5. Prepare the Local Database result set. If the table exists but it must be recreated it is dropped. In this case and if it doesn't exist, a new table is created.
 6. Create a new XML document using the template stored into the Data Package.

7. Create a XML node to add the records to be added or updated in the device (rowSet) and a XML node to add the records to be deleted in the device (deletedRowSet).
8. Execute the query to get the data from the Local Database table.
9. Create the statements to be used to delete, update or insert data into the local table.
10. Process data record by record:
 - If both result sets are at the end or empty the process ends.
 - If the Data Center result set is at the end or empty but the Local Database result set has more records, all these records are deleted from the Local Database result set and added to the XML deletedRowSet node.
 - If the Local Database result set is at the end or empty but the Data Center result set has more records, all these following records are inserted into the Local Database result set and added to the XML rowSet node.
 - If both result sets are not empty or are at the end, the primary key is compared between the both current records. If they have the same primary key, every field is compared. If there is any difference the Local Database record is updated and his data is add to the XML rowSet node. Then both result sets go to the next record
 - When the primary key comparison is done, if the primary key of the Data Center record is different from the Local Database record two different course of action take place:
 - If the Data Center record is more advanced, the Local Database record must be deleted and added to the XML deletedRowSet node. Then the Local Database result set goes to the next record.
 - If the Local Database record is more advanced, the Data Center record must be added to the Local Database result set and added to the XML rowSet node. Then the Data Center result set goes to the next record.
11. Once data process finish the XML rowSet and deletedRowset nodes are added to the XML document. If during the preparation of the Local Database result set was established that a new database and/or new table must be created on the device now these attributed are set on the XML document.
12. The XML document is serialized to a local file, added to the Package to be sent to the device and the local file is deleted.
13. All the connections, statements and result set are close.
14. If during the data process something fails, all the changes in the Local Database are roll back.

- The Packager finishes its work.

The exceptions are handled in this ways:

- The work continues and the exception is logged.
- The work is aborted and the exception logged.
- The work continues, the exception is logged and a mail is sent with the exception explained.

- The work is aborted, the exception is logged and a mail is sent with the exception explained.

Data types support

Four classes of data types are already supported: integer (signed four bytes), floats (double precision), string (255 bytes long) and long text (32 000 bytes long). It is restricted by the device database manager system. JDBC data types are mapped to this classification. Only when the original data cannot fit into a device field, the data is truncated to the length can be handled by the device. That is the case of LONGVARCHAR and VARCHAR coming for some server DBMSs

Data Piping Configuration

The Data Piping Configuration allows the user to configure Data Package, Database Source and Filtering Component definitions to be used by the Data Piping Process.

Login

The Data Piping configuration requires a valid logon to utilize the functionality that it provides. This is achieved through the login page, which prompts for a username, password and domain. This is compared against information in the LDAP provider.

Package management

Provides access to list, create, modify and delete the Data Packages definitions.

List

Displays a list of all Data Packages definitions with these attributes:

- Name.
- Datasource.
- Databasename.
- Table.
- Filtering Component
- XML.

Name, Datasource, Filtering Component and XML are links.

Click on the Name allows editing the Data Package definition. If a Data Package is in use by any application, it is not allowed to edit the Data Package definition so the name is not showed as a link.

Click on Datasource allows editing the Database Source definition whose name is listed.

Click on Filtering Component allows editing the Filtering Component specification whose name is listed.

Clicks on xml to show the XML configuration file for this particular Data Package.

This option also has a menu bar at the end of the list. It has the option *Add* to allows to add a new Data Package definition.

Show XML file.

Displays the Data Package definition configuration file. Accessed from the Package management – List xml link.

Create

The user can create a new Data Package definition. Starts by clicking the option *Add* in the Package Management – List

During this wizard like process, is possible to navigate forward and back among the pages or discard all by clicking *Back*, *Next* and *Discard* on the menu bar.

Select a database source.

The first step is to select a database source, that have been created using Database Sources.

Once a source is selected from a list, the details about are showed:

- Driver. JDBC driver used by the Data Piping to create database connections. Specific for DBMS.
- URL. URL address of the database to be accessed.
- User. A valid user name with rights to access the data to be piped.

The user must enter a valid password for this user in order to proceed with the following step.

Select tables.

Shows two lists with the tables can be selected on the left and the selected tables on the right. The system tables in the database are not listed. Between the lists are four options to select or deselect items, multiple selections is allowed. At least one table must be selected. Each time a user changes the table selection, the fields, relations and primary key list are reset.

Select fields.

Shows two lists with the fields can be selected on left and the selected fields on the right. The fields belongs to the selected tables in the preview page and only fields whose data types can be processed by the Data Piping. Each time a user changes the fields selection, the relations and primary key list are reset. Between the lists are four options to select or deselect items, multiple selections is allowed. At least one field must be selected.

Relate tables.

When more than one table is selected, this page is used to relate all the selected tables. Shows two lists with the fields can be selected for the relations. Includes all the fields that belongs to selected tables and can be used as primary key. A third list below shows all defined relations. Three options allow setting a relation between the fields selected on the first two lists, deleting one relation or deleting all of them.

Select primary key.

A Data Package definition must have a primary key in order to be possible compare and process the data from the source and the local image.

Shows two lists with the fields that can be selected on the left and the selected fields on the right. The fields belong to the selected tables in the preview page and only fields whose data types can be processed by the Data Piping. Each time a user changes the table selection, the field lists are reset. Between the lists are four options to select or deselect items, multiple selections are allowed. At least one field must be selected.

Select filtering component, package name, database name and table name.

This page allows setting the component to be used as filter by the Data Piping by selecting one in a list of the defined previously in Filtering Components. The default selection is not to use any component and it is a valid choice.

In addition, the user must set the database and table names. The combination of a database and table name must be unique across all the Data Packages definitions.

Finally, a unique name is given to the Data Package.

By clicking on XML Document the user goes to the last step.

Show XML file.

Shows the XML document generated using the information gathered from the user. By clicking on *Accept* the document is stored in the configuration database and the list of all already defined Data Package is showed, including the new one.

Modify

The user can modify an existing Data Package definition. Starts by clicking a name in the Package Management – List

During this wizard-like process, it is possible to navigate forward and back among the pages or discard all changes by clicking *Back*, *Next* and *Discard Changes* on the menu bar.

Show XML file.

Shows the XML document, generated using the information gathered from the user or stored in the configuration database. By clicking on *Accept Changes* the document is updated in the configuration database and the list of all already defined Data Package is showed.

By clicking on *Edit* starts the modification of the Data Package definition.

Select a database source.

The first step is to select a database source, that have been created using Database Sources. The default is that is already used by this Data Package. If the user selects a different one, the tables, fields, relations and primary key will be reset.

Once a source is selected from a list, the details about are showed:

- Driver: JDBC driver used by the Data Piping to create database connections. Specific for DBMS.

- URL: URL address of the database to be accessed.
- User: A valid user name with rights to access the data to be piped.

The user must enter a valid password for this user in order to proceed with the following step.

Edit tables selection.

Shows two lists with the tables can be selected on the left and the selected tables on the right. The system tables in the database are not listed. Between the lists are four options to select or deselect items, multiple selections is allowed. At least one table must be selected. Each time a user changes the table selection, the fields, relations and primary key list are reset.

Edit fields selection.

Shows two lists with the fields can be selected on left and the selected fields on the right. The fields belongs to the selected tables in the preview page and only fields whose data types can be processed by the Data Piping. Between the lists are four options to select or deselect items, multiple selections is allowed. Each time a user changes the fields selection, the relations and primary key list are reset. At least one field must be selected.

Edit tables relations.

When more than one table is selected, this page is used to relate all the selected tables. Shows two lists with the fields can be selected for the relations. Includes all the fields that belongs to the selected table and can be used as primary key. A third list below shows all defined relations. Three options allow setting a relation between the fields selected on the first two lists, deleting one relation or deleting all of them.

Edit primary key.

A Data Package definition must have a primary key in order to be possible compare and process the data from the source and the local image.

Shows two lists with the fields can be selected on left and the selected fields on the right. The fields belongs to the selected tables in the preview page and only fields whose data types can be processed by the Data Piping. Each time a user changes the table selection, the field lists are reset. Between the lists are four options to select or deselect items, multiple selections is allowed. At least one field must be selected.

Edit filtering component, package name, database name and table name.

This page allows change the component to be used as filter by the Data Piping by selecting one in a list of the defined previously in Filtering Components. The default selection is not to use any component and it is a valid choice.

In addition, the user can change the database and table names. The combination of a database and table name must be unique across all the Data Packages definitions.

By clicking on XML Document, the user goes to the return to show the XML including now all the modifications did by the user.

Database Sources

Allows the developer to configure the parameters to access databases to be used as data source for processing.

List

Display a list of all of the Database Sources currently defined with these attributes:

- Name: Identifier of the Database Source.
 - Driver: JDBC driver used by the Data Piping to create database connections. Specific for DBMS.
 - URL: URL address of the database to be accessed.
 - User: User. A valid user name with rights to access the data to be piped.
- By clicking on a name, the user can edit a Database Source definition.
By clicking on *Add*, it is possible to create a new definition.

Create

Allows configuring a new Database Source by setting up the following attributes:

- Name
- Driver
- URL
- User
- Password

The name must be unique. All fields are required. The password must be valid for this User and his must have access rights to the data to be piped.

It is possible to save the new definition by click on *Accept* or return to the list without store this information by click on *Discard*.

Before storing the information, the Database Source is validated against the actual database.

Modify

Allows configuring an existing Database Source by updating the following attributes:

- Driver
- URL
- User
- Password

The name is showed but it may not to be edited. All fields are required. The password must be valid for this User and his must have access rights to the data to be piped.

It is possible to save the new definition by click on *Accept* or return to the list without store this information by click on *Discard*.

Before storing the information, the Database Source is validated against the actual database.

Filtering Components

The developer can use this option to set up a component to filter the data to be processed by the Data Piping.

List

Display a list of all of the Filtering Components currently defined with these attributes:

- Name: Identifier of the Filtering Components.
- URL: URL address used by the Data Piping to invoke the SOAP RPC Router.
- Object Name: Component SOAP identifier.
- Implementation: Allows to access components using the Microsoft or Apache implementations of SOAP.
- Description: A short description of the component.

By clicking on a name, the user can edit a Filtering Components definition.

By clicking on *Add*, it is possible to create a new definition.

Create

Allows configuring a new Database Source by setting up the following attributes:

- Name
- URL
- Object Name
- Implementation
- Description

The name must be unique. All fields but Description are required.

It is possible to save the new definition by click on *Accept* or return to the list without store this information by click on *Discard*.

Modify

Allows configuring an existing Database Source by updating the following attributes:

- Driver
- URL
- Object Name
- Implementation
- Description

The name is showed but it may not to be edited. All fields but Description are required.

It is possible to save the new definition by click on *Accept* or return to the list without store this information by click on *Discard*.

Logout

This feature will log the current user out of the Data Piping configuration. Access to any feature of the console after logging out will require logging back in to the system with a valid account.

Data Package Definition XML

The Data Package Definition XML document defines which data will be getting from the Enterprise Data Center by set the Datasource that provides access to the data, list the tables and fields, the relations among the tables and the Filtering Component that gives the developer tailored condition employed by the Data Process. In addition sets the

database and table where the data will be stored on the client device and includes additional instruction about database and table creation on the device.

The root element is *package* with the attributes: *name* (identifier), *database*, *table* (database and table names on the device), *datasource* (datasource identifier).

Has the following child elements:

- The first child element is *filteringComponent*, the Filtering Component identifier.
- The second child element is *tables*. Has at least one or more *table* child element with *name* attribute (identifier of a table in the Enterprise Data Center).
- The third child element is *fields*. Has at least one or more *field* child element with *name* and *table* attributes (identifiers of a field on a table in the Enterprise Data Center).
- The fourth child element is *relations*. Has zero, one or more *relation* child elements with *stmt* attribute (relation between two of the tables).
- The last child element is *order*. Has at least one or more *orderBy* child element with *name* and *table* attributes (identifiers of a field on a table in the Enterprise Data Center).

Data Process Result XML

This XML document is the basic result of the Data Piping Process. Includes the instruction to create or delete a new database or table and the data to be added, modified or deleted from the device database. It is self-contained, includes not only the data but the table definition too.

The root element is *database* with the attributes *name* (identifier) and optionally *create* and *delete* to instruct the device to create or delete the specified database.

Has only one child element *table* with the attributes *name* (identifier) and optionally *create* and *delete* to instruct the device to create or delete the specified table.

- *Table* has a child element *fields* with the attribute *count* (fields on the table). *Fields* has one or more *field* child element with *maxLength* (maximum length of the field), *mayBeNull* (0 if not, 1 if it can be null), *name* (name of the field), *number* (sequence order of the field), *pk* (1 if it is a primary key, 0 if not), *type* (1 is character string, 2 is an integer value, 3 is a float number, 4 is a long text).
- Has also a child element *rowSet* that has zero or more *row* child elements. Each *row* has as many *fieldData* child elements as fields the result set has. *FieldData* has a *number* attribute (sequence order of the field) and stores the value of the field. The data in the *rowSet* will be inserted or updated on the client.
- The last child element is *deletedRowSet* that has zero or more *deletedRow* child elements. Each *row* has as many *fieldData* child elements as fields the result set has. *FieldData* has a *number* attribute (sequence order of the field) and stores the value of the field. The data in the *rowSet* will be deleted on the client database.

Glossary

Scheduler

Scheduling Process

Description

Allow programming the process of data packages on regular bases. Candidates are those data packages whose update time are well known or doesn't change very often, like the system tables. It improves the synchronization process by just processing the data, needs to be updated at the time of the device connection.

The Data Packages must be configured as scheduled and set the schedule in order to be processed by the scheduler.

In addition, the interval between scheduler executions should be set. The default interval is 60 minutes.

Init Scheduler

Starts the scheduling of data preparation tasks to be executed at a fixed interval. The starting time is calculated by adding the interval to the beginning of the current date until have the next time before the moment the Scheduler starts.

Creates a timer that schedule the data preparation processes at a fixed rate based on the interval and the starting time calculated as was explained before.

Data Preparation Process

Each time a new Data Preparation Task is started by the timer, a new Data Preparation Process is executed:

- Purges all the inactive Data Packages, by selecting from the configuration database and the local database those packages that have some references and data on the local database but don't belongs to any application in use by any group or don't have any scheduling configuration being set as scheduled. Then all the local database references and data are deleted and created Packager files to be sent to the devices with the necessary instructions to update the client databases.
- Selects from the configuration database which Devices and Data Packages will be processed at this time ordering the result by Devices.
- Creates a new Packager file for each new Device to be processed.
- For each Data Packages to be synchronized, process the data from the Enterprise Data Center and creates and add the correspondent XML document to the same package to be sent to the device. It includes:
 1. Invokes via SOAP the Filtering Component to get the filtering condition to be added at the WHERE condition of statement used to inquire the Data Center.
 2. Using the Datasource attribute of the Data Package, create a Datasource definition and use that to open a connection to the Data Center.
 3. Create a connection to the Local Database where resides an image of the data already has the device.
 4. Execute the query to get the data to be processed from the Data Center.

5. Prepare the Local Database result set. If the table exists but it must be recreated it is dropped. In this case and if it doesn't exist, a new table is created.
6. Create a new XML document using the template stored into the Data Package.
7. Create a XML node to add the records to be added or updated in the device (rowSet) and a XML node to add the records to be deleted in the device (deletedRowSet).
8. Execute the query to get the data from the Local Database table.
9. Create the statements to be used to delete, update or insert data into the local table.
10. Process data record by record:
 - If both result sets are at the end or empty the process ends.
 - If the Data Center result set is at the end or empty but the Local Database result set has more records, all these records are deleted from the Local Database result set and added to the XML deletedRowSet node.
 - If the Local Database result set is at the end or empty but the Data Center result set has more records, all these following records are inserted into the Local Database result set and added to the XML rowSet node.
 - If both result sets are not empty or are at the end, the primary key is compared between the both current records. If they have the same primary key, every field is compared. If there is any difference the Local Database record is updated and his data is add to the XML rowSet node. Then both result sets go to the next record
 - When the primary key comparison is done, if the primary key of the Data Center record is different from the Local Database record two different course of action take place:
 If the Data Center record is more advanced, the Local Database record must be deleted and added to the XML deletedRowSet node. Then the Local Database result set goes to the next record.
 If the Local Database record is more advanced, the Data Center record must be added to the Local Database result set and added to the XML rowSet node. Then the Data Center result set goes to the next record.
11. Once data process finish the XML rowSet and deletedRowset nodes are added to the XML document. If during the preparation of the Local Database result set was established that a new database and/or new table must be created on the device now these attributed are set on the XML document.
12. The XML document is serialized to a local file, added to the Package to be sent to the device and the local file is deleted.
13. All the connections, statements and result set are close.
14. If during the data process something fails, all the changes in the Local Database are roll back.
- The Packager finishes its work.

Stop Scheduler

Stops the Scheduler by cancel the timer.

Glossary

D R A F T

04/27/2001

XOOM Development Strategies

During the architectural and development phase of XOOM several influencing factors in the design were confronted. Each of these factors was analyzed and solutions were developed to protect the purity and business value of the final product. Following are the different categories in which influencing factors had potential impacts and solutions implemented.

Influencing Factors

1. The cost of a Data Base engine for each handheld mobile and embedded device could increase each of the device cost substantially.
2. Memory on the handheld mobile and embedded devices was limited and costly to increase relative to the cost of the devices.
3. None of the handheld mobile and embedded devices had implemented JNI (Java Native Interface).
4. Staying agnostic to the WEB Application Servers in the market to avoid compatibility and deployment limitations with potential customers.
5. Device data independence to remove any need to mandate data be represented or structured without restrictions.
6. Maintain an architecture agnostic to the two major WEB services architectures provided by Microsoft and the JAVA environments.
7. Allow for the SQL Database to be present on the device, but avoid data replication as a method for synchronization. By adapting this architecture we would avoid the seat cost of licensing on the Server side.
8. Allow for the Application Business Logic to be represented in the Server in any development language.

Solutions

The architecture of the XOOM product was adapted to meet the factors influencing our design. Following is our strategy to address each of the influencing factors.

Our strategy for factors 1, 2 and 3 was addressed by initially supporting the Windows environment using the WIN 32 API, thereby minimizing the requirement for memory size on the device and lack of support for the Java Native Interface. We also implemented our device Data Base using the Windows CE packaged CDB database included as part of the Windows CE offering.

Item 4 was addressed by developing what is referred to as the XOOM container and pooling mechanism to avoid using a commercially available WEB application Server

semi-proprietary architectures. It must be noted our analysis showed we would have only used 6% – 8% of any WEB Application Server capabilities. In addition, we found several WEB Application Servers not yet fully compliant with the most current J2EE standards being implemented in XOOM. This solution resulted in significant savings in the cost of the XOOM Framework to the customer and Abaco.

Item 5 was addressed by separating the data elements from the application, thereby allowing the data component to reside and be shared at the device level by multiple applications.

Item 6 was addressed by implementing in our architecture decision points to determine the environment and take the appropriate process paths to address the difference in their implementation of standards. The dominant architectures addressed were Microsoft and JAVA environments.

Item 7 was addressed by allowing the SQL Database to be present on the device, but avoid data replication as a method for synchronization. By adapting this architecture we would avoid the seat cost of licensing on the Server side. This was accomplished with our TxSync module as the sole user of the Server Data Base not the end users and maintaining a separation of the transaction and back end data.

Item 8 was accomplished by deploying all our WEB services in a JAVA environment. In addition, developing an ISAPI to front end this environment for Microsoft implementations and allow Business Objects deployed in COM+, Enterprise Java Beans (EJB), Java Servlets and others to be accessed via Soap requests. This implementation will allow our customers to leverage their investments in business and application logic investments.

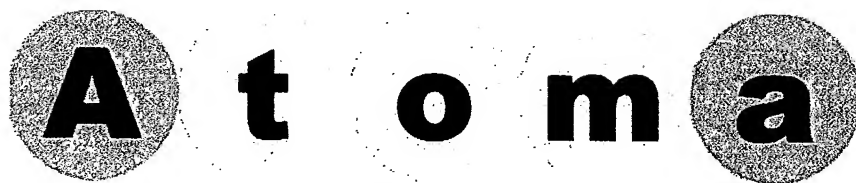
Mission

Provide a framework that enables developers to create complex enterprise applications for mobile devices using popular and established programming technologies; enables system administrators to manage (configure, monitor, deploy to...) mobile devices using the web infrastructure; provide enterprises with a highly scalable system capable of addressing thousands of mobile users.

Framework Requirements

1. Applications developed with framework must be able to target online (network connection available), offline (no network connection available), and semi-connected (network connection is intermittently available) environments.
2. Applications developed with framework must be able to use Internet based protocols to communicate when connected to a network.
3. Applications developed with framework must be able to access native interfaces (OS API functions & Device Specific APIs - Scanners, printers, pen capture...) of client devices.
4. Applications developed with framework must be able to communicate securely when connected to a network.
5. Framework must allow applications to target client device operating system of choice.
6. Elements of framework residing on mobile device must minimize size and memory, processor and battery usage.
7. Framework must allow applications to access logic components that reside on a remote computer independent of remote computer operating system or programming language used to develop logic components.
8. Framework must utilize network as efficiently as possible.
9. System administrator(s) must be able to centrally manage and administer all mobile devices and system parameters.
10. System administrator(s) must be able to remotely manage and administer all mobile devices and system parameters.
11. System must provide a mechanism to synchronize business transactions between a mobile device and a central system.
12. System must guarantee the integrity of the information transmitted in the synchronization process.
13. System must provide a mechanism to push user-specific data to a the mobile device.
14. Elements of system residing on server must support the most popular enterprise server operating systems and hardware architectures.

15. Elements of system residing on server must scale to support large numbers of mobile users simultaneously.
16. System must enable remote deployment of device framework elements, enterprise applications, and user-specific data without presence of any system specific elements on device.



Mobile Services

Atoma Technical Overview

White Paper

Abstract

This paper gives information technology professionals a technical overview of the features provided by Abaco's Atoma framework. The features of the framework include a scalable architecture for mobile application development and deployment and a management system for mobile devices utilizing disparate operating systems.

© 2001 Abaco PR, Inc. All rights reserved.

The information contained in this document represents the current view of Abaco Mobile on the issues discussed as of the date of publication. Because Abaco Mobile must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Abaco, and Abaco cannot guarantee the accuracy of any information presented after the date of publication.

This white paper is for informational purposes only. ABACO MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Other product and company names mentioned herein may be the trademarks of their respective owners.

**Abaco • 1100 Northmeadow Pkwy. Suite 150 • Roswell, GA 30076-4960 • USA
2001**

CONTENTS

INTRODUCTION 1
Architecture Brief 1

ETUP AND DEPLOYMENT 3

CONSOLE 4
User Management 4
Monitoring & Configuration 4
Security Administration 4
Application Management 4

USER AUTHENTICATION..... 7

SECURE AUTHENTICATION..... 8

DATA PIPING..... 9

TX SYNC..... 11

DEVICE FRAMEWORK..... 13

CONCLUSION..... 14

ADDITIONAL RESOURCES..... 15

INTR DUCTION

Nobody likes to sit still. As enterprises reach for greater efficiencies in all areas of operation the general consensus is that in order to achieve improved efficiency each arm of the enterprise must make the best use of time.

Consider a beverage sales representative who visits customers on a daily route. This representative creates orders for products as requested by each customer, keeps notes regarding sales trends as information is gathered from each customer, and gives information regarding promotions and new products to each customer. How does the sales rep keep informed of new promotions offered by the beverage company? When placing an order how does the rep provide accurate feedback as to when the order will be shipped, print invoices, scan products that require reorders to speed information gathering and ensure accuracy? How can the rep automatically read information collected in vending machines that are on the sales route?

Now lets examine the requirements of a solution for the beverage sales representative's real enterprise needs:

- There are 5000 sales representatives working on routes and the beverage company will not depend on one device vendor so the application must support multiple device brands.
- The sales representatives perform most of their duties off-site, in areas where network coverage is not available.
- The orders created by the sales representatives must use information stored in enterprise information systems and must be created while applying centralized business rules established for sales orders. These business rules have already been implemented using a number of disparate technologies including CORBA, EJB's and COM+.
- Due to an ultra competitive environment between beverage companies the system must ensure that communications containing sales information are secured and that only authorized sales representatives are able to access the application and corporate resources.

In order for the beverage company to be successful in delivering this solution, the resources at the beverage company must be able to focus on their core competencies: the business rules of the company and the application functionality required. In order for the solution to be able to grow with the growing needs of the company and it's employees, the solution must take advantage of open protocols and standards to ensure its longevity.

To achieve enterprise goals and to meet system requirements such as the ones outlined above and those that may come in the future a robust and open mobile architecture is needed and that architecture is Atoma.

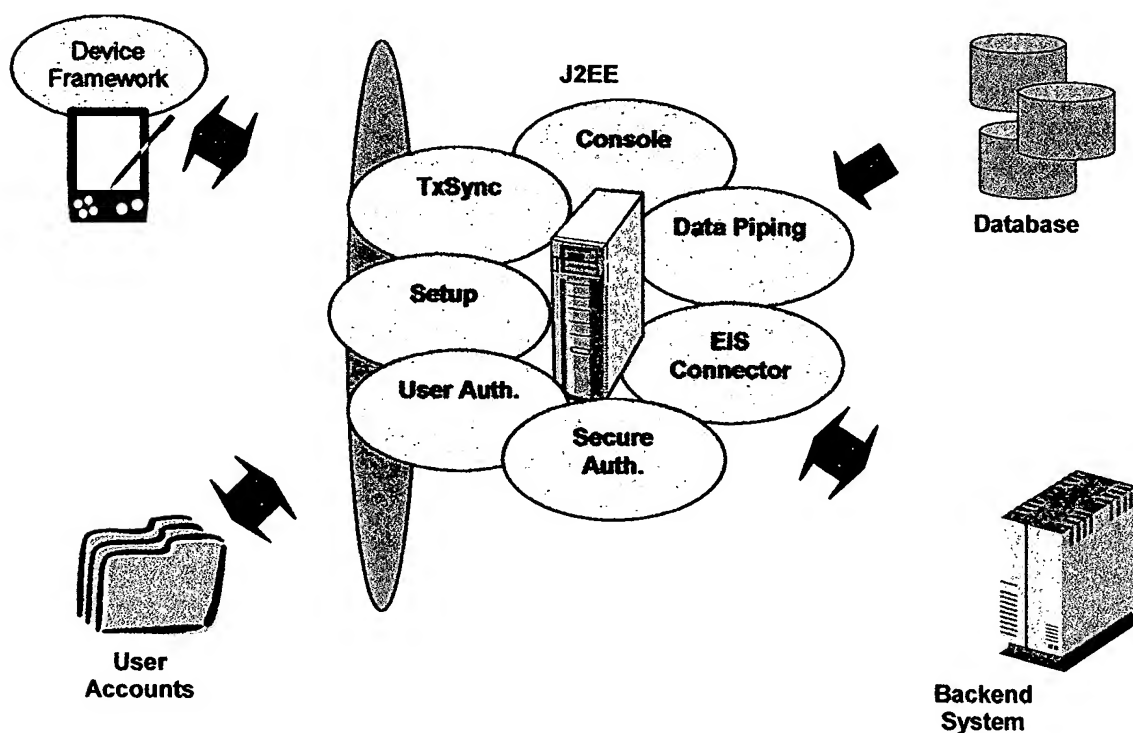
Architecture Brief

The Atoma architecture is designed to support large numbers of mobile devices, provide services independent of the operating system and platform and provide speed and

efficient usage of the limited resources found in mobile devices. The server-side architecture is based in J2EE, works in concert with any application server on the market and allows the use of any programming object model for logic encapsulation and business rule reuse.

The client-side architecture is based in the native languages supported by the device operating systems to achieve a small memory footprint, efficient power consumption, and ability to interface with any peripheral supported by the device.

Figure 1: ATOMA Architecture



SETUP AND DEPLOYMENT

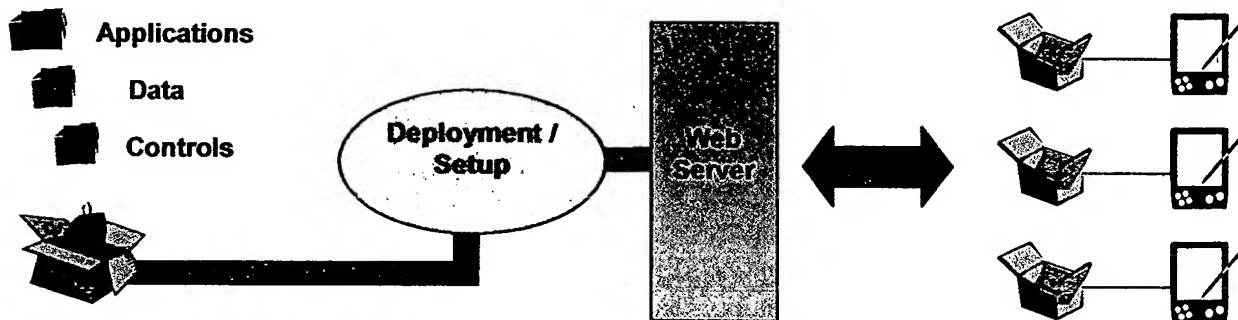
One of the primary goals of the Atoma framework is to streamline the process of initializing and configuring mobile devices. The setup and configuration of devices is typically the most mundane time consuming and costly process associated with the management of a mobile system. Due to the volatile nature of embedded device usable memory, the setup and configuration cycle is typically repeated each time the device battery is discharged or the device reverts back to factory settings because of a reset operation or condition. By automating the setup process, the management of a mobile system is greatly simplified and significant cost savings are realized due to the reduction in human intervention necessary to initialize a device.

The mobile device setup process is primarily achieved through a browser. Using the web browser found on today's mobile devices, a user simply navigates to the HTTP URL associated with an Atoma server. Once the user is authenticated, the device being used is automatically identified and the device framework components appropriate for the device are downloaded. These components are packaged in a compressed self-extracting module that installs the included components and performs a number of additional steps to complete the setup process. During these additional steps the applications assigned to the user are installed and the data and database structures for the user are downloaded to the device.

While most of today's mobile devices are equipped with a browser, settings for a network connection typically must be configured before the browser becomes useful. To overcome this obstacle, a Setup Card utility is provided where using compact flash and eventually Smart Cards, custom network configurations can be prepared and saved on a Setup Card. When inserted into the device this card automatically configures the connection settings of the device allowing the browser-based setup to proceed seamlessly.

Atoma's automated device setup process provides an elegant and efficient solution to one of the more difficult problems facing any mobile device management system. The benefits of this automation include a drastic reduction in the time required for device initialization and a corresponding increase in the overall user-friendliness and availability of the system

Figure 2 Setup process



C NSOLE

A key feature of the Atoma system is the capability to centrally manage an entire network of client devices and to also manage an Atoma installation from the same environment. These capabilities are provided by the AtomaConsole - a web based application which empowers a system administrator to remotely configure and monitor client devices and also manage server-side settings and configuration options.

User Management

The Console avoids the administrative nightmare of creating and managing multiple user databases by seamlessly integrating with user management systems already in use by an enterprise. After specifying an LDAP* compliant repository where user accounts are stored, the Console can be used to organize imported users into groups.

*Microsoft Windows NT Security Accounts Manager also supported

Monitoring & Configuration

As users synchronize using Atoma's TxSync process, information describing the status and settings of a device are reported to an Atoma server. This information can be viewed for any device at any time through the Console allowing system administrators to monitor the devices to establish user trends and to preempt device related issues. The Console also enables configuration of client devices where the applied settings are realized during the TxSync process. Through device configuration, system administrators can take control of numerous settings on the client device such as power management settings, network connection settings, device display settings, and much more.

Security Administration

Coupled with the framework's support for Secure Sockets Layer and 128-Bit Encryption, a complete Certificate Authority is provided to enable issue and revocation of Digital Client Certificates to system users. The settings for this Certificate Authority and the settings for Digital Client Authentication are all managed through the Console.

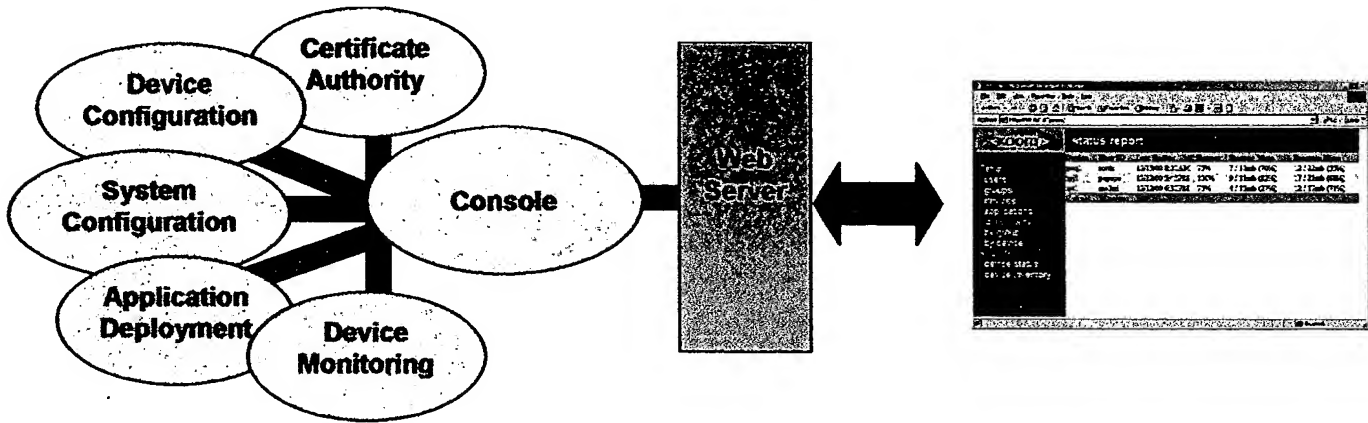
Application Management

An Atoma application can take on many forms: an executable, a set of HTML forms and pages combined with scripting technologies. Regardless of the form an application takes, it is ultimately delivered to an enterprise user's device to allow that user to perform his or her daily tasks. Using the Console administrators and developers can centrally deploy applications to all users of the Atoma system. Applications can be assigned to groups of users (as defined above in User Management). Updated versions of applications can also be deployed such that the application will be updated on all devices having different versions.

The Console is the central administration and configuration tool for all Atoma modules and processes. Due to the adaptability of the Atoma framework, there are several options and settings that can be used to customize the framework for distinct enterprise computing environments. The Console consolidates the myriad of options and tools into

a manageable, user-friendly web application.

Figure 3 Console



USER AUTHENTICATION

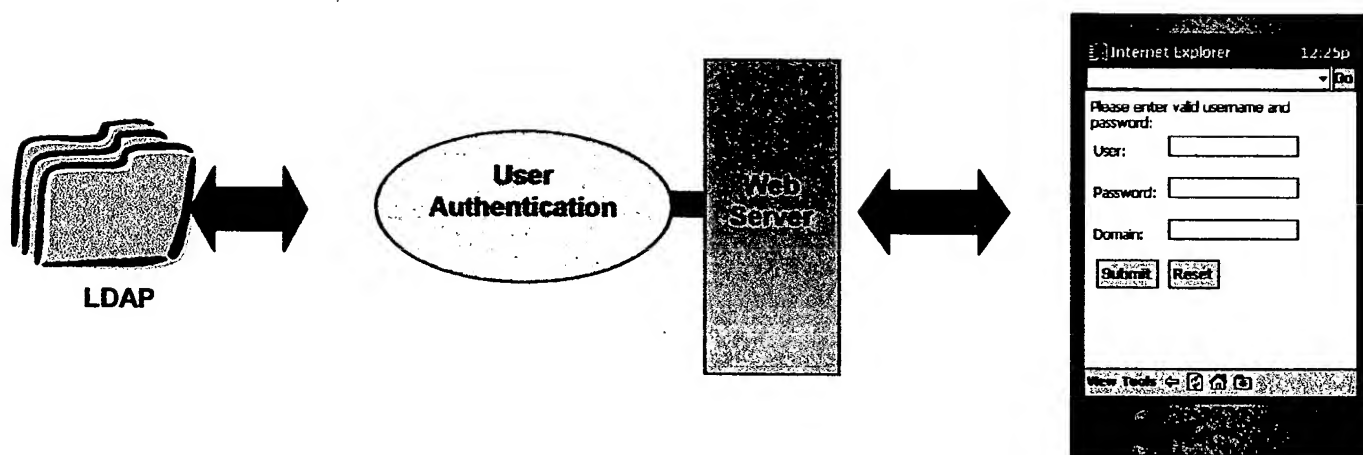
User Authentication is the process of verifying the credentials of individuals who try to access a system's resources. User authentication provides the basic foundation for any secure software or networking system. There are several excellent systems in use today for the management of user accounts and the storage of user related information. Instead of duplicating the services provided by enterprise class user account management systems and causing system administrators to have to maintain duplicate data in separate user repositories, the Atoma framework allows you to take advantage of the capabilities provided by an existing user management system by reading and validating any user related information against such an existing repository.

The Atoma framework will interface with any LDAP compliant user account system and also includes support for the Microsoft Windows™ NT based security and accounts management systems. This interface occurs on two levels. First, in order to successfully administer an Atoma system, information about the users of the client devices in the system is required. This information is always pulled automatically from a designated master user account repository alleviating an administrator from having to create and manage users separately for the Atoma system.

The second level of interaction between an Atoma system and an enterprise user account system is at the time of validation of a user's credentials. User credentials are always validated by the enterprise user account management system. Any time that a user logon of some sort is required by the system, the validation of the user's credentials is performed by the master user account system. By adhering to this model, Atoma ensures that user information remains truly centralized and avoids several possible security weaknesses that might be introduced when user information, such as passwords, is duplicated in multiple systems.

The Atoma system's use of User Authentication ensures that access to any enterprise resources available through the system is only granted after the requesting party's credentials have been centrally authorized. Tight integration with existing user account management systems ensures that administrators do not duplicate user management efforts and that users do not need to keep track of yet another username and password combination.

Figure 4 User Authentication



SECURE AUTHENTICATION

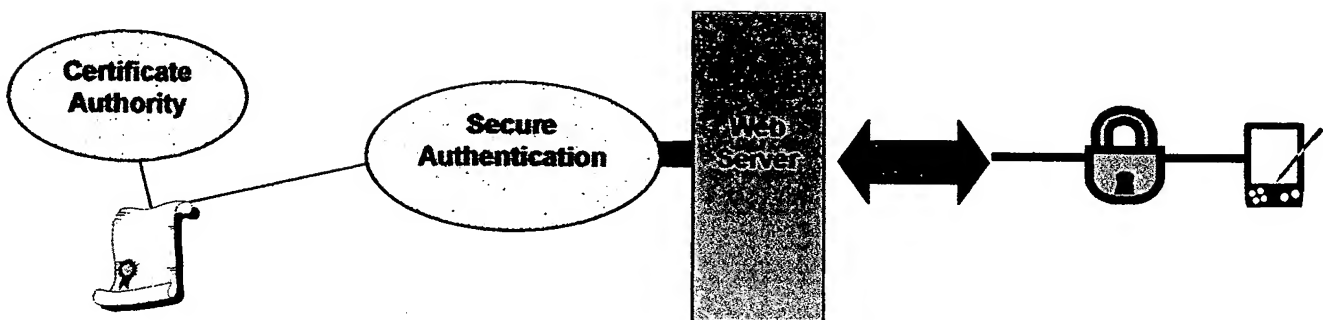
Secure Authentication is the process by which users requesting server services are granted or denied access during an otherwise secure communication exchange. Communications exchanged between device and server can be secured with 128-bit encryption using Secure Sockets Layer (SSL) and Digital Server Certificates. Secure Authentication adds Digital Client Certificates to these technologies to add verification of the identity of the party requesting a server's services and resources before granting access to those services and resources.

The Atoma system's implementation of Secure Authentication provides automatic assignment and deployment of digital certificates to users. Once a digital client certificate has been deployed to a user it is no longer necessary to prompt the user for logon credentials to validate his or her identity. The Atoma Device Framework takes care of presenting the user's client certificate when secure authentication is requested allowing the server to positively identify the user before granting access to a requested resource.

A number of server-side services are provided to support the Secure Authentication implementation. One such service is the Certificate Authority that manages the digital client certificates in use on the system. The Certificate Authority allows the creation and revocation of client certificates and provides a resource for validation of existing client certificates. An HTTP request filter is also provided to read certificates as they are submitted by client devices and to interface with the Certificate Authority to verify the validity of any given certificate.

Secure Authentication provides a way to digitally authenticate users to ensure that access to enterprise services is strictly regulated based on a user's credentials. The use of digital client certificates provides an added level of security to SSL communication exchanges that could otherwise possibly allow unauthorized use of enterprise resources.

Figure 5 Secure Authentication



DATA PIPING

Data Piping enables mobile users to receive individualized information to be stored locally on a client device.

The Data Piping process begins with the Data Piping Designer. This web-based tool allows application architects and developers to specify the data elements, originating from centralized enterprise data sources, which are required for their mobile application(s) and how these elements will be structured on the target device. These centralized enterprise data sources can be any combination of databases from the major database management system products in use today, such as DB2, Informix, Oracle, SQL Server and Sybase.

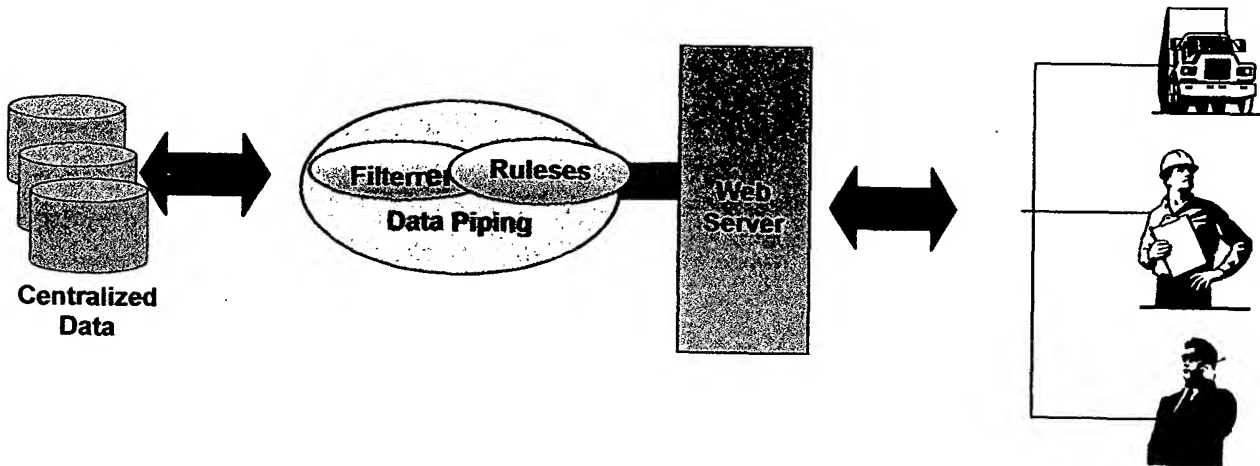
The Data Piping Designer essentially allows application designers to create application relevant subsets of the information stored centrally in an enterprise by graphically creating queries that can take into account specific user characteristics. While this functionality can solve many problems for an enterprise, Atoma Data Piping allows an even further customization of the process. The system allows developers to supplement the queries created in the Designer with logic implemented in software objects to handle complex data dependencies and tasks that require processing which cannot be implemented through queries alone.

Data Piping is closely integrated with the TxSync process. During the TxSync process the data and data structures that are relevant to the user conducting the synchronization are delivered to the user's device in a platform independent XML-based format. This XML-based format allows freedom of choice to mix and match database management systems on the server with database systems on the device, allowing enterprises to select best of breed solutions instead of being tied to any single vendor.

The actual preparation of data and data structures for a specific user and piping instruction set can happen in one of two ways: by schedule and/or during synchronization. When preparation is set to occur by schedule, the data destined for a device user is extracted from the centralized data sources at a specified time (or times) on a regular cycle. When preparation occurs during synchronization, the queries to extract data and the accompanying logic are executed during the TxSync process to provide the ultimate in flexibility and to ensure that only the most up-to-date data available is transferred to the user's device.

Atoma Data Piping provides an engine capable of providing enterprise users with local data stores for information access while disconnected from a network and keeping these local data stores up-to-date with ever-changing data in centralized enterprise data sources. Data piping allows developers to create and manage data structures, specify data associations and compliment these structures and associations with logic to provide an intelligent method of providing enterprise users with only the information they need, when it is needed.

Figure 6 Data Piping



TX SYNC

Abaco's TxSync is a unique process by which disconnected devices enabled with the Atoma Device Framework accomplish synchronization with enterprise network resources. The primary goals of the TxSync architecture are to provide a scalable mechanism for simultaneous synchronization of large numbers of mobile users and to provide a model for reporting information from a client device to an enterprise resource that ensures the integrity of the information transmitted.

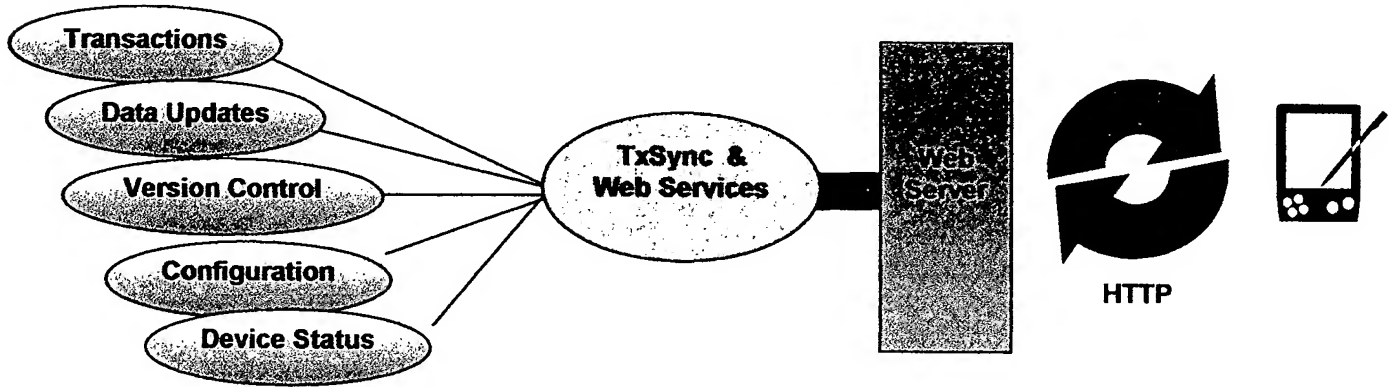
TxSync achieves its goals by combining SOAP (Simple Object Access Protocol) and a completely asynchronous architecture for maximum efficiency and reliability during the device synchronization process. TxSync adheres to the rules of fully self-contained transactions from the ground up. TxSync's fundamental concept is to use SOAP for all data traffic from the device to the enterprise resource. In this way the complexity, inherent pitfalls, and inevitable data collisions encountered in raw data synchronization or data replication systems is avoided. An added benefit of TxSync's use of SOAP for data uploads is that it allows enterprises to take control of data validation and processing rules without being tied to any single platform or system.

TxSync provides several features which ensure and protect the integrity of enterprise data as it is reported. Communications between the device and the TxSync server are completely asynchronous allowing straightforward recovery from inevitable breaks in network connectivity while the TxSync process is executing. The server modules of the TxSync system uniquely identify each synchronization to ensure that, in the case of a system error, data and calls to enterprise objects are not erroneously duplicated. The server modules also manage their state using non-volatile memory, such that in the case of a complete hardware failure, an in-process synchronization can resume where it was interrupted after a server is restarted.

During the TxSync process several tasks are completed in one robust operation: application transactions are reported to the server, changes to application database structures and data are prepared depending on the device-user and his organizational role, current status of the device is reported, version control of applications and system components is performed and system configuration changes are conveyed to the device. During the TxSync process potentially large sets of data are compressed to make best use of the available network connectivity and usage of the network is minimized by packaging needed information and data into single packages to eliminate the need for multiple calls and responses between the device and the server.

The TxSync process provides Atoma systems with a fully integrated, robust mechanism for asynchronous information exchange between a client device and the enterprise network. The TxSync process enables mobile users to spend as much time as needed untethered to a network or operating in areas where coverage is not available with the assurance that the information created by their work will be reliably reported to their enterprise and their device will be kept up-to-date with the latest corporate information and settings applied by the system administrator.

Figure 7 TxSync



DEVICE FRAMEWORK

The Atoma device frameworks are designed to provide developers with a foundation to create robust enterprise applications. The fundamental services provided by a device framework are usage of web protocols for application communication, simplified and platform independent access to data capture peripherals, support for offline and online applications, efficient usage of device resources to conserve power usage, and automated application management, deployment and configuration.

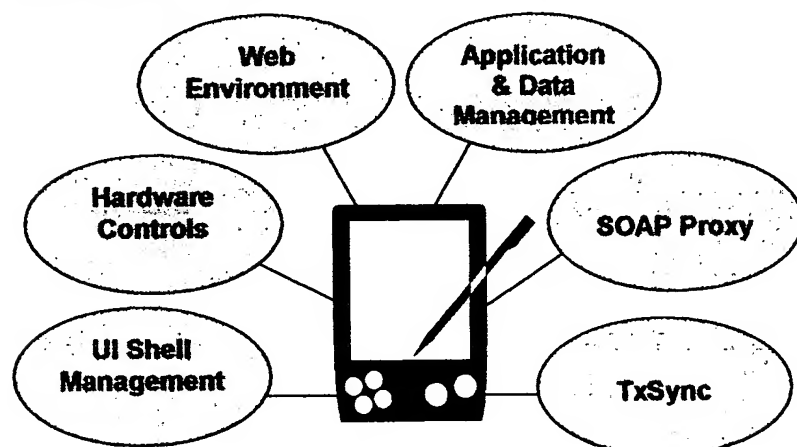
Each Atoma device framework targets a particular operating system platform. Today ATOMA is delivered with the Windows CE device framework that targets Pocket PC and HPC 2000 devices. Future device frameworks will target Mobile Linux and Epoc-32. For Windows CE, the device framework consists of a set of application programming controls, executable files, and supporting tools.

Device frameworks provide an application programming interface that supports embedded device application developed using web languages (HTML and ASP on CE), rapid application development languages (eMbedded VB on CE) and core supported languages (C/C++ on CE). Enterprises benefit from the device frameworks by reducing application development cycles, enjoying functionally rich applications, and having freedom of choice between different device manufacturers with minimal or no application portability issues.

The automated services provided by an Atoma device framework present enterprises with an unparalleled opportunity to streamline operations and ensure that a mobile solution is operating efficiently and reliably. Many of these automated services are managed by the TxSync process to create the ultimate in flexibility between online and offline modes of operation. Services such as application version control, data piping, device configuration, and secure authentication give system administrators a level of control very rare in most mobile solutions.

To create a mobile application that has the capabilities required by Today's enterprises, two main ingredients are necessary: the correct mobile device for the target environment and a strong application development and management framework to support the desired application. The Atoma device frameworks deliver on the second ingredient while allowing you to choose freely from any of the first ingredient available on the market today and tomorrow.

Figure 8 Device Framework



C NCLUSI N

Delivering a mobile solution is an inherently complex endeavor. While technological advances and the proliferation of new technologies have opened new possibilities, very few products have offered a cohesive solution to the requirements and desires of the mobile enterprise.

By delivering automated Setup and Deployment, enterprise User Authentication, Secure Authentication and communications, Transactional-based synchronization, complete system and application version control, remote device configuration and monitoring, and an intelligent data deployment architecture, Atoma puts enterprises in a position to realize the goals outlined in a mobile strategy and to reap the resulting benefits.

ADDITI NAL RESOURCES

Abaco Mobile Web site:

<http://www.abacomobile.com/atoma/>

Java 2 Enterprise Edition Website:

<http://java.sun.com/j2ee/>

Microsoft Pocket PC Web site:

[http:// www.microsoft.com/mobile/pocketpc/default.asp](http://www.microsoft.com/mobile/pocketpc/default.asp)



Xml Open Object Model

XOOM Technical Overview

White Paper

Abstract

This paper gives information technology professionals a technical overview of the features provided by Abaco's XOOM framework. The features of the framework include a scalable architecture for mobile application development and deployment and a management system for mobile devices utilizing disparate operating systems.

© 2001 Abaco PR, Inc. All rights reserved.

The information contained in this document represents the current view of Abaco Mobile on the issues discussed as of the date of publication. Because Abaco Mobile must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Abaco, and Abaco cannot guarantee the accuracy of any information presented after the date of publication.

This white paper is for informational purposes only. ABACO MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Abaco • 1100 Northmeadow Pkwy, Suite 150 • Roswell, GA 30076-4960 • USA
2001

C NTENTS

INTRODUCTION.....1
Architecture Brief 1

SETUP AND DEPLOYMENT3

CONSOLE.....4
User Management 4
Monitoring & Configuration 4
Security Administration 4
Application Management 4

USER AUTHENTICATION7

SECURE AUTHENTICATION8

DATA PIPING9

TX SYNC11

DEVICE FRAMEWORK13

CONCLUSION14

ADDITIONAL RESOURCES15

INTRODUCTION

Nobody likes to sit still. As enterprises reach for greater efficiencies in all areas of operation the general consensus is that in order to achieve improved efficiency each arm of the enterprise must make the best use of time.

Consider a beverage sales representative who visits customers on a daily route. This representative creates orders for products as requested by each customer, keeps notes regarding sales trends as information is gathered from each customer, and gives information regarding promotions and new products to each customer. How does the sales rep keep informed of new promotions offered by the beverage company? When placing an order how does the rep provide accurate feedback as to when the order will be shipped, print invoices, scan products that require reorders to speed information gathering and ensure accuracy? How can the rep automatically read information collected in vending machines that are on the sales route?

Now let's examine the requirements of a solution for the beverage sales representative's real enterprise needs:

- There are 5000 sales representatives working on routes and the beverage company will not depend on one device vendor so the application must support multiple device brands.
- The sales representatives perform most of their duties off-site, in areas where network coverage is not available.
- The orders created by the sales representatives must use information stored in enterprise information systems and must be created while applying centralized business rules established for sales orders. These business rules have already been implemented using a number of disparate technologies including CORBA, EJB's and COM+.
- Due to an ultra competitive environment between beverage companies the system must ensure that communications containing sales information are secured and that only authorized sales representatives are able to access the application and corporate resources.

In order for the beverage company to be successful in delivering this solution, the resources at the beverage company must be able to focus on their core competencies: the business rules of the company and the application functionality required. In order for the solution to be able to grow with the growing needs of the company and its employees, the solution must take advantage of open protocols and standards to ensure its longevity.

To achieve enterprise goals and to meet system requirements such as the ones outlined above and those that may come in the future a robust and open mobile architecture is needed and that architecture is XOOM.

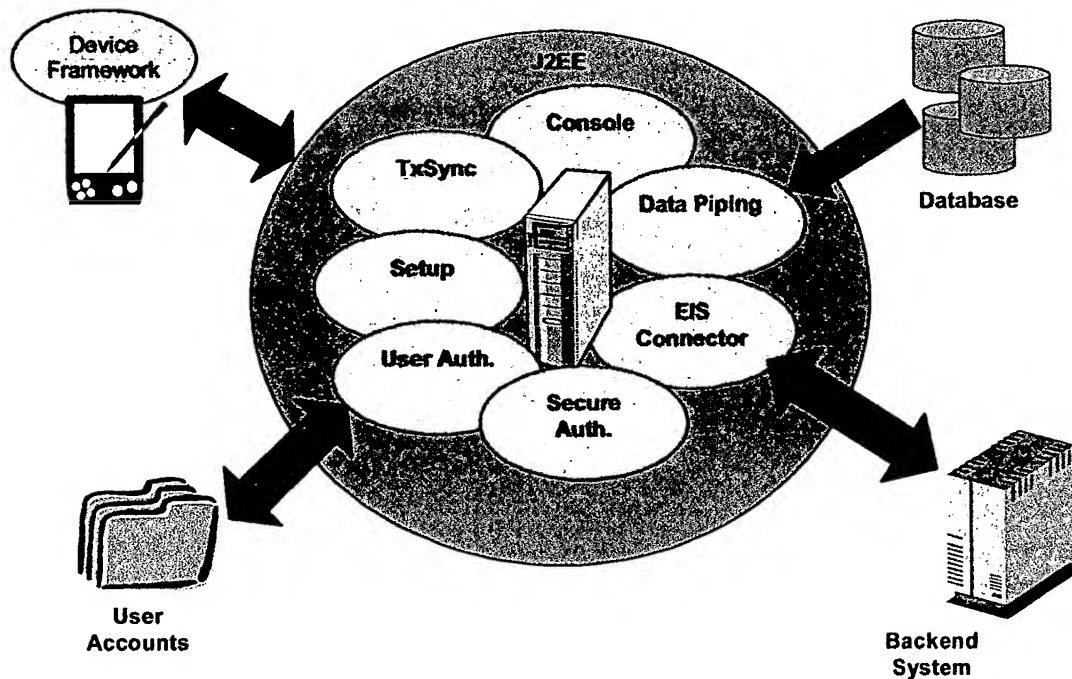
Architecture Brief

The XOOM architecture is designed to support large numbers of mobile devices, provide services independent of the operating system and platform and provide speed and

efficient usage of the limited resources found in mobile devices. The server-side architecture is based in J2EE, works in concert with any application server on the market and allows the use of any programming object model for logic encapsulation and business rule reuse.

The client-side architecture is based in the native languages supported by the device operating systems to achieve a small memory footprint, efficient power consumption, and ability to interface with any peripheral supported by the device.

Figure 1: XOOM Architecture



SETUP AND DEPLOYMENT

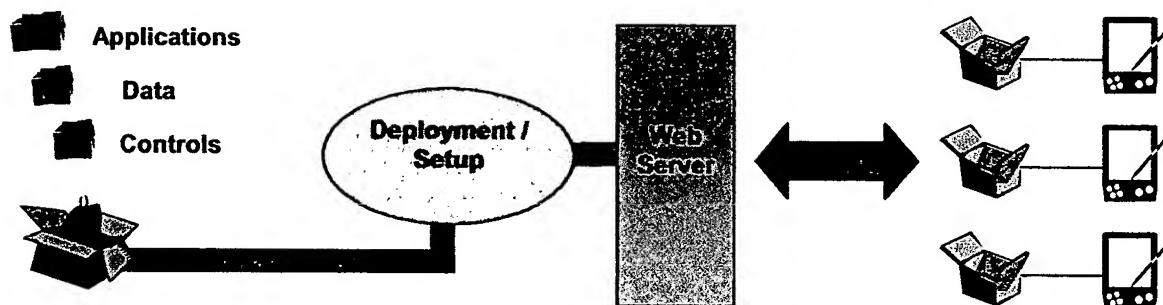
One of the primary goals of the XOOM framework is to streamline the process of initializing and configuring mobile devices. The setup and configuration of devices is typically the most mundane time consuming and costly process associated with the management of a mobile system. Due to the volatile nature of embedded device usable memory, the setup and configuration cycle is typically repeated each time the device battery is discharged or the device reverts back to factory settings because of a reset operation or condition. By automating the setup process, the management of a mobile system is greatly simplified and significant cost savings are realized due to the reduction in human intervention necessary to initialize a device.

The mobile device setup process is primarily achieved through a browser. Using the web browser found on today's mobile devices, a user simply navigates to the HTTP URL associated with a XOOM server. Once the user is authenticated, the device being used is automatically identified and the device framework components appropriate for the device are downloaded. These components are packaged in a compressed self-extracting module that installs the included components and performs a number of additional steps to complete the setup process. During these additional steps the applications assigned to the user are installed and the data and database structures for the user are downloaded to the device.

While most of today's mobile devices are equipped with a browser, settings for a network connection typically must be configured before the browser becomes useful. To overcome this obstacle, a Setup Card utility is provided where using compact flash and eventually Smart Cards, custom network configurations can be prepared and saved on a Setup Card. When inserted into the device this card automatically configures the connection settings of the device allowing the browser-based setup to proceed seamlessly.

XOOM's automated device setup process provides an elegant and efficient solution to one of the more difficult problems facing any mobile device management system. The benefits of this automation include a drastic reduction in the time required for device initialization and a corresponding increase in the overall user-friendliness and availability of the system

Figure 2 Setup process



A key feature of the XOOM system is the capability to centrally manage an entire network of client devices and to also manage a XOOM installation from the same environment. These capabilities are provided by the XOOM Console - a web based application which empowers a system administrator to remotely configure and monitor client devices and also manage server-side settings and configuration options.

User Management

The Console avoids the administrative nightmare of creating and managing multiple user databases by seamlessly integrating with user management systems already in use by an enterprise. After specifying an LDAP* compliant repository where user accounts are stored, the Console can be used to organize imported users into groups.

*Microsoft Windows NT Security Accounts Manager also supported

Monitoring & Configuration

As users synchronize using XOOM's TxSync process, information describing the status and settings of a device are reported to a XOOM server. This information can be viewed for any device at any time through the Console allowing system administrators to monitor the devices to establish user trends and to preempt device related issues. The Console also enables configuration of client devices where the applied settings are realized during the TxSync process. Through device configuration, system administrators can take control of numerous settings on the client device such as power management settings, network connection settings, device display settings, and much more.

Security Administration

Coupled with the framework's support for Secure Sockets Layer and 128-Bit Encryption, a complete Certificate Authority is provided to enable issue and revocation of Digital Client Certificates to system users. The settings for this Certificate Authority and the settings for Digital Client Authentication are all managed through the Console.

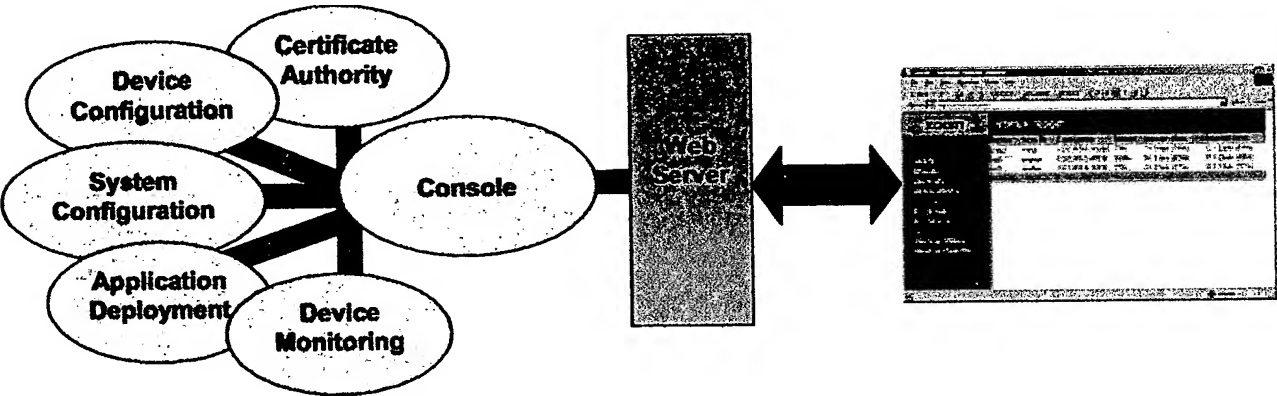
Application Management

A XOOM application can take on many forms: an executable, a set of HTML forms and pages combined with scripting technologies. Regardless of the form an application takes, it is ultimately delivered to an enterprise user's device to allow that user to perform his or her daily tasks. Using the Console administrators and developers can centrally deploy applications to all users of the XOOM system. Applications can be assigned to groups of users (as defined above in User Management). Updated versions of applications can also be deployed such that the application will be updated on all devices having different versions.

The Console is the central administration and configuration tool for all XOOM modules and processes. Due to the adaptability of the XOOM framework, there are several options and settings that can be used to customize the framework for distinct enterprise computing environments. The Console consolidates the myriad of options and tools into

a manageable, user-friendly web application.

Figure 3 Console



USER AUTHENTICATION

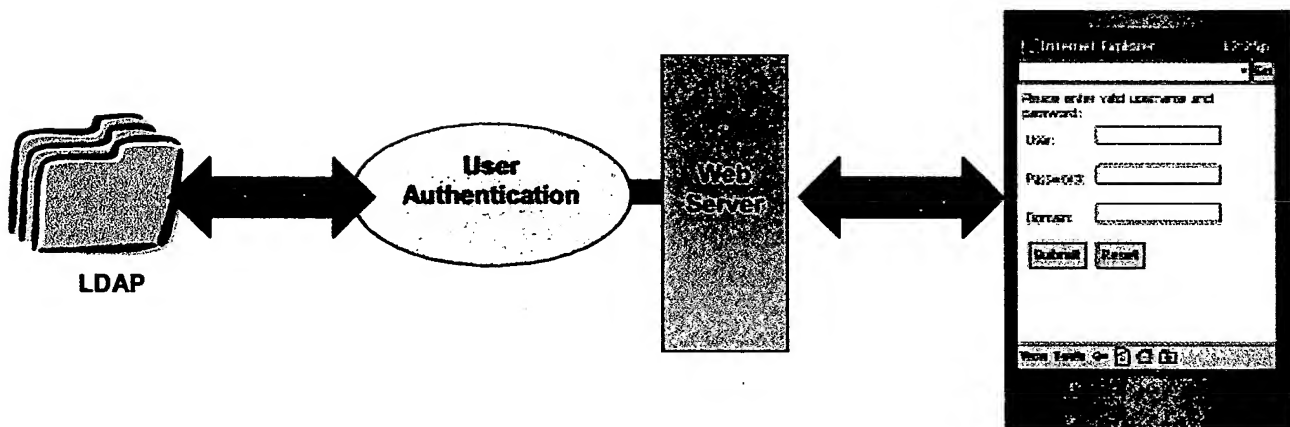
User Authentication is the process of verifying the credentials of individuals who try to access a system's resources. User authentication provides the basic foundation for any secure software or networking system. There are several excellent systems in use today for the management of user accounts and the storage of user related information. Instead of duplicating the services provided by enterprise class user account management systems and causing system administrators to have to maintain duplicate data in separate user repositories, the XOOM framework allows you to take advantage of the capabilities provided by an existing user management system by reading and validating any user related information against such an existing repository.

The XOOM framework will interface with any LDAP compliant user account system and also includes support for the Microsoft Windows™ NT based security and accounts management systems. This interface occurs on two levels. First, in order to successfully administer a XOOM system, information about the users of the client devices in the system is required. This information is always pulled automatically from a designated master user account repository alleviating an administrator from having to create and manage users separately for the XOOM system.

The second level of interaction between a XOOM system and an enterprise user account system is at the time of validation of a user's credentials. User credentials are always validated by the enterprise user account management system. Any time that a user logon of some sort is required by the system, the validation of the user's credentials is performed by the master user account system. By adhering to this model, XOOM ensures that user information remains truly centralized and avoids several possible security weaknesses that might be introduced when user information, such as passwords, is duplicated in multiple systems.

The XOOM system's use of User Authentication ensures that access to any enterprise resources available through the system is only granted after the requesting party's credentials have been centrally authorized. Tight integration with existing user account management systems ensures that administrators do not duplicate user management efforts and that users do not need to keep track of yet another username and password combination.

Figure 4 User Authentication



SECURE AUTHENTICATION

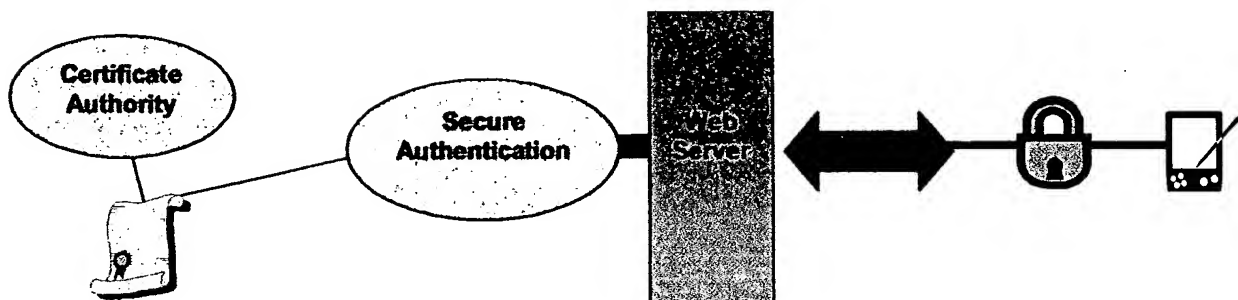
Secure Authentication is the process by which users requesting server services are granted or denied access during an otherwise secure communication exchange. Communications exchanged between device and server can be secured with 128-bit encryption using Secure Sockets Layer (SSL) and Digital Server Certificates. Secure Authentication adds Digital Client Certificates to these technologies to add verification of the identity of the party requesting a server's services and resources before granting access to those services and resources.

The XOOM system's implementation of Secure Authentication provides automatic assignment and deployment of digital certificates to users. Once a digital client certificate has been deployed to a user it is no longer necessary to prompt the user for logon credentials to validate his or her identity. The XOOM Device Framework takes care of presenting the user's client certificate when secure authentication is requested allowing the server to positively identify the user before granting access to a requested resource.

A number of server-side services are provided to support the Secure Authentication implementation. One such service is the Certificate Authority that manages the digital client certificates in use on the system. The Certificate Authority allows the creation and revocation of client certificates and provides a resource for validation of existing client certificates. An HTTP request filter is also provided to read certificates as they are submitted by client devices and to interface with the Certificate Authority to verify the validity of any given certificate.

Secure Authentication provides a way to digitally authenticate users to ensure that access to enterprise services is strictly regulated based on a user's credentials. The use of digital client certificates provides an added level of security to SSL communication exchanges that could otherwise possibly allow unauthorized use of enterprise resources.

Figure 5 Secure Authentication



DATA PIPING

Data Piping enables mobile users to receive individualized information to be stored locally on a client device.

The Data Piping process begins with the Data Piping Designer. This web-based tool allows application architects and developers to specify the data elements, originating from centralized enterprise data sources, which are required for their mobile application(s) and how these elements will be structured on the target device. These centralized enterprise data sources can be any combination of databases from the major database management system products in use today, such as DB2, Informix, Oracle, SQL Server and Sybase.

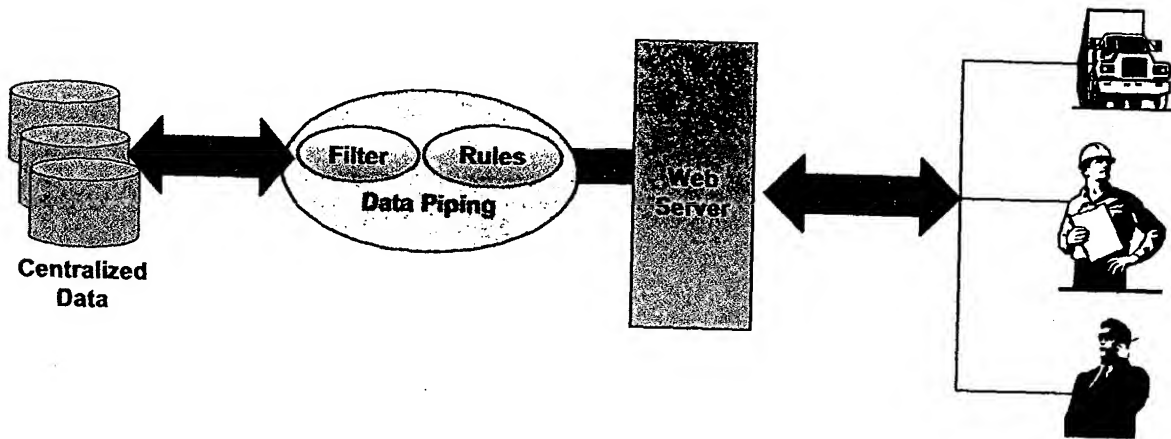
The Data Piping Designer essentially allows application designers to create application relevant subsets of the information stored centrally in an enterprise by graphically creating queries that can take into account specific user characteristics. While this functionality can solve many problems for an enterprise, XOOM Data Piping allows an even further customization of the process. The system allows developers to supplement the queries created in the Designer with logic implemented in software objects to handle complex data dependencies and tasks that require processing which cannot be implemented through queries alone.

Data Piping is closely integrated with the TxSync process. During the TxSync process the data and data structures that are relevant to the user conducting the synchronization are delivered to the user's device in a platform independent XML-based format. This XML-based format allows freedom of choice to mix and match database management systems on the server with database systems on the device, allowing enterprises to select best of breed solutions instead of being tied to any single vendor.

The actual preparation of data and data structures for a specific user and piping instruction set can happen in one of two ways: by schedule and/or during synchronization. When preparation is set to occur by schedule, the data destined for a device user is extracted from the centralized data sources at a specified time (or times) on a regular cycle. When preparation occurs during synchronization, the queries to extract data and the accompanying logic are executed during the TxSync process to provide the ultimate in flexibility and to ensure that only the most up-to-date data available is transferred to the user's device.

XOOM Data Piping provides an engine capable of providing enterprise users with local data stores for information access while disconnected from a network and keeping these local data stores up-to-date with ever-changing data in centralized enterprise data sources. Data piping allows developers to create and manage data structures, specify data associations and compliment these structures and associations with logic to provide an intelligent method of providing enterprise users with only the information they need, when it is needed.

Figure 6 Data Piping



TX SYNC

Abaco's TxSync is a unique process by which disconnected devices enabled with the Xoom Device Framework accomplish synchronization with enterprise network resources. The primary goals of the TxSync architecture are to provide a scalable mechanism for simultaneous synchronization of large numbers of mobile users and to provide a model for reporting information from a client device to an enterprise resource that ensures the integrity of the information transmitted.

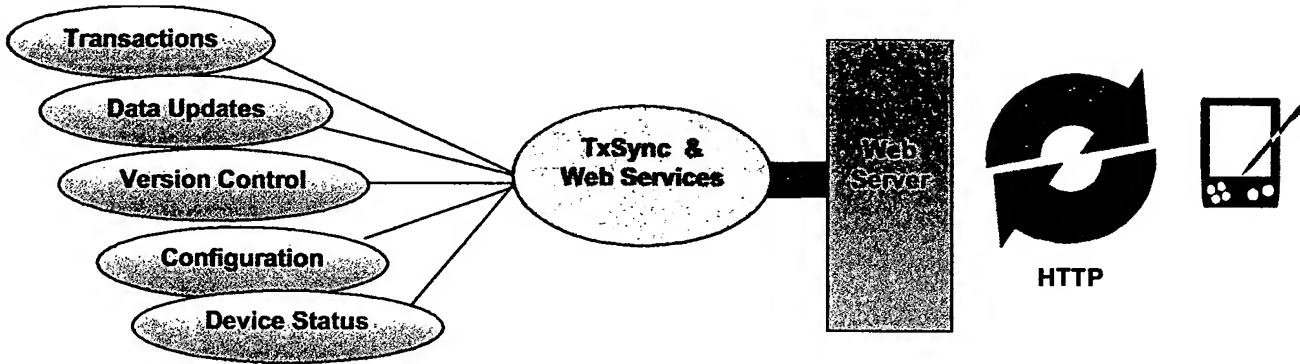
TxSync achieves its goals by combining SOAP (Simple Object Access Protocol) and a completely asynchronous architecture for maximum efficiency and reliability during the device synchronization process. TxSync adheres to the rules of fully self-contained transactions from the ground up. TxSync's fundamental concept is to use SOAP for all data traffic from the device to the enterprise resource. In this way the complexity, inherent pitfalls, and inevitable data collisions encountered in raw data synchronization or data replication systems is avoided. An added benefit of TxSync's use of SOAP for data uploads is that it allows enterprises to take control of data validation and processing rules without being tied to any single platform or system.

TxSync provides several features which ensure and protect the integrity of enterprise data as it is reported. Communications between the device and the TxSync server are completely asynchronous allowing straightforward recovery from inevitable breaks in network connectivity while the TxSync process is executing. The server modules of the TxSync system uniquely identify each synchronization to ensure that, in the case of a system error, data and calls to enterprise objects are not erroneously duplicated. The server modules also manage their state using non-volatile memory, such that in the case of a complete hardware failure, an in-process synchronization can resume where it was interrupted after a server is restarted.

During the TxSync process several tasks are completed in one robust operation: application transactions are reported to the server, changes to application database structures and data are prepared depending on the device-user and his organizational role, current status of the device is reported, version control of applications and system components is performed and system configuration changes are conveyed to the device. During the TxSync process potentially large sets of data are compressed to make best use of the available network connectivity and usage of the network is minimized by packaging needed information and data into single packages to eliminate the need for multiple calls and responses between the device and the server.

The TxSync process provides XOOM systems with a fully integrated, robust mechanism for asynchronous information exchange between a client device and the enterprise network. The TxSync process enables mobile users to spend as much time as needed untethered to a network or operating in areas where coverage is not available with the assurance that the information created by their work will be reliably reported to their enterprise and their device will be kept up-to-date with the latest corporate information and settings applied by the system administrator.

Figure 7 TxSync



DEVICE FRAMEWORK RK

The XOOM device frameworks are designed to provide developers with a foundation to create robust enterprise applications. The fundamental services provided by a device framework are usage of web protocols for application communication, simplified and platform independent access to data capture peripherals, support for offline and online applications, efficient usage of device resources to conserve power usage, and automated application management, deployment and configuration.

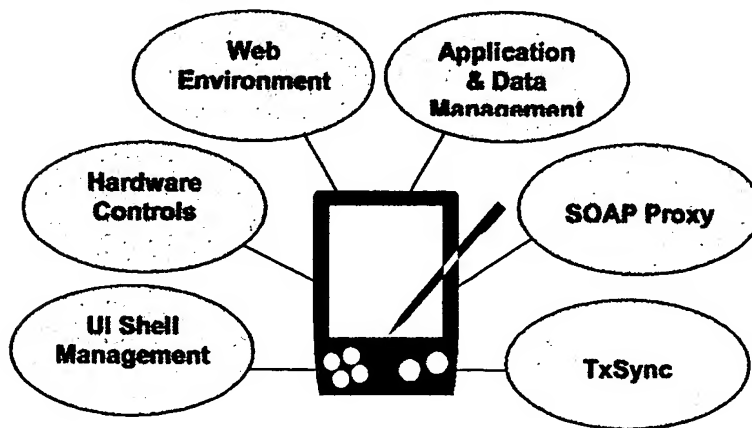
Each XOOM device framework targets a particular operating system platform. Today XOOM is delivered with the Windows CE device framework that targets Pocket PC and HPC 2000 devices. Future device frameworks will target Mobile Linux and Epoc-32. For Windows CE, the device framework consists of a set of application programming controls, executable files, and supporting tools.

Device frameworks provide an application programming interface that supports embedded device application developed using web languages (HTML and ASP on CE), rapid application development languages (eMbedded VB on CE) and core supported languages (C/C++ on CE). Enterprises benefit from the device frameworks by reducing application development cycles, enjoying functionally rich applications, and having freedom of choice between different device manufacturers with minimal or no application portability issues.

The automated services provided by a XOOM device framework present enterprises with an unparalleled opportunity to streamline operations and ensure that a mobile solution is operating efficiently and reliably. Many of these automated services are managed by the TxSync process to create the ultimate in flexibility between online and offline modes of operation. Services such as application version control, data piping, device configuration, and secure authentication give system administrators a level of control very rare in most mobile solutions.

To create a mobile application that has the capabilities required by Today's enterprises, two main ingredients are necessary: the correct mobile device for the target environment and a strong application development and management framework to support the desired application. The XOOM device frameworks deliver on the second ingredient while allowing you to choose freely from any of the first ingredient available on the market today and tomorrow.

Figure 8 Device Framework



CONCLUSION

Delivering a mobile solution is an inherently complex endeavor. While technological advances and the proliferation of new technologies have opened new possibilities, very few products have offered a cohesive solution to the requirements and desires of the mobile enterprise.

By delivering automated Setup and Deployment, enterprise User Authentication, Secure Authentication and communications, Transactional-based synchronization, complete system and application version control, remote device configuration and monitoring, and an intelligent data deployment architecture, XOOM puts enterprises in a position to realize the goals outlined in a mobile strategy and to reap the resulting benefits.

ADDITIONAL RESOURCES

Abaco Mobile Web site:

<http://www.abacomobile.com/xoom/>

Java 2 Enterprise Edition Website:

<http://java.sun.com/j2ee/>

Microsoft Pocket PC Web site:

[http:// www.microsoft.com/mobile/pocketpc/default.asp](http://www.microsoft.com/mobile/pocketpc/default.asp)